سلسلة كن أسدا للإبداع

BE LION CREATIVITY

سبيلك إلى تعلم لغة

C#.NET

خالد السعداني



سبيلك إلى تعلم لغة C#.Net

من إعداد : خالد السعداني

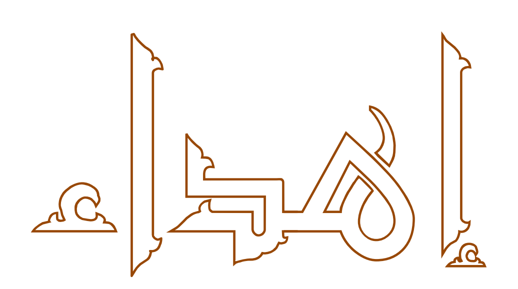


"يا أيها الذين آمنوا اتقوا الله و قولوا قولا سديدا. يصلح لكم أعمالكم و يغفر لكم ذنوبكم ومن يطع الله و رسوله فقد فاز فوزا عظيما"

الأحزاب : 70 و 71







إلى الوالدين أولا أطال الله عمرهما و جعلنا بارين لهما،
وإلى البلاد الكريمة : المغرب
وكل بلاد المسلمين و إلى كل المسلمين والمسلمات



اللهم اجعله عملا خالصا لوجهك







الإسم الكامل : خالد السعداني

الهاتف: 0673-07-51-05

من مواليد 18/05/1989 بمدينة الفقيه بن صالح ، المغرب

تقني متخصص في التنمية المعلوماتية

الكفاءات المهنية : البرمجة ب :

vb.net,c#.net,C++,C,Java(J2ee)

وبالنسبة للويب :

Asp.net,PHP,Html,JavaScript,Ajax,CSS,jQuery

أما قواعد البيانات :

MS Access,MS SQL Server,MySQL



بسم الله توكلت على الله





بسم الله الرحمان الرحيم

أما بعد،

فقد يكون من المثير للجدل أن تسمى هذه السلسلة –التي نأمل أن تتضافر الجهود لتطويرها و النهوض بها– بهذا الاسم الباعث على الحماس و على روح التحدي في وقت أصبحت فيه هممنا تساطح الأرض بعدما كانت تناطح أعناق الجوزاء، وفي وقت أصبحنا نلعب نحن دور المتتبع للحدث من دون أن نشارك فيه والأخر دور الملقي و الفاعل.

لا أود أن أتقمص دور المصلح لأن كفاءاتي المتواضعة لا تسمح لي، و لأن معارفي بعلم السوسيولوجيا تكاد تكون محدودة إن لم أقل منعدمة، ففقط من باب الغيرة و السعي إلى رد الاعتبار للهوية العربية و المسلمة ، ألفت هذه السلسلة، راجيا من الله العلي القدير أن يجعلها عونا لكل طالب و باحث.

وبعيدا عن العنوان، أود التنبيه إلى أن كون السلسلة مجانية و متوفرة على الانترنت، وكل ما أرومه من ورائها دعوة صالحة لي و لوالدي و لكل المسلمين وأن ينفع الله بها كل من قرأ منها حرفا.

خالد السعداني



عند وجود أي ملاحظة، المرجو مراسلتي عبر :

Khalid_Essaadani@Hotmail.Fr



الفهرس


BE LION
arabic formation

تقديم

بسم الله الرحمان الرحيم،



قبل خوض غمار البرمجة بلغة السي شارب، أود أن أبدأ بشرح بعض الأشياء التي أرى أنه من الواجب أن تكون ضمن معارف كل مبرمج يسعى الى أن يكون أسدا في هذه اللغة، بعجالة فهذه اللغة ظهرت نسخها التجريبية بشكل متتابع منذ سنة 2000 إلى أن تم إصدار أول نسخة رسمية سنة 2002 ضمن إطار العمل Framework 1.0 هذا الأخير يعتبر بمثابة الطبقة القاعدية اللازمة لكي تعمل السي شارب على نظام الويندوز . تماما مثل الآلة الإفتراضية بالنسبة للغة الجافا.

لهذا فكل اللغات التي تشتغل تحت إطار العمل Framework لها تقريبا نفس المردود فالإختيار بين هاته اللغات لاتحكمه قوة و إمكانيات اللغة وإنما يبقى الإختيار مجرد ميول و ارتياح للغة على حساب أخرى فالنتائج التي ستصل إليها باستعمال فئات الفيجوال بسيك تستطيع تحقيقها أيضا بواسطة السي شارب أو بغيرها .

عرفت لغة السي شارب تطورا ملحوظا مع تقدم الوقت لتظهر النسخة 2.0 و 3.0 و 3.5 لتصل حاليا إلى 4.0 مع إطار العمل 4.0.

إنها بحق لغة تستحق التعلم ، نظرا لسلاستها و سهولتها و كفاءاتها العالية و اعتمادها على البرمجة الشيئية التي سنراها في فصل لاحق إن شاء الله ضمن سلسلة "كن أسدا" فكن أسدا بحق، و سم الله ثم غص معي في بحر السي شارب لننتقي منه أبهى الدرر.



أساسيات لغة السي شارب

1. أساسيات لغة السي شارب:

1.1 المتغيرات

من الشائع أن دور الذاكرة الحية في جهاز الحاسوب هو حفظ القيم للتعامل معها من خلال البرنامج المعني، نفس الشيء ينطبق أيضا على مفهوم المتغيرات غير أن هذه الأخيرة تحمل أسماء ليسهل التعامل معها، بالإضافة إلى نوعها. فمثلا إذا أردنا أن نقوم بعمل برنامج يقوم بجمع عددين فيلزمنا أن نحجز في الذاكرة الحية حيزين لمتغيرين رقميين ثم نقوم بعد ذلك بإعطائهما قيما لحساب مجموعهما، فبذلك سيكون البرنامج كالتالي :

```
Integer a ;
Integer  b ;
A=3 ;
B=5 ;
Write(a+b) ;
```

وحتى أوضح لك الرؤية من أجل التمهيد لكتابة البرنامج بلغة السي شارب، فهذا الكود الزائف يقوم بالإعلان على متغيرين a و b نوعهما رقمي ، أي قمنا بحجز مكانين في الذاكرة، بعد ذلك أعطينا هذين المتغيرين قيما و في الأخير قمنا بطباعة النتيجة. إن لم تتمكن من استيعاب مفهوم المتغيرات فأنصحك بإعادة قراءة هذه الفقرة قبل المرور إلى أول برنامج بلغة السي شارب.

1.2 بنية برنامج بلغة السي شارب :

باستعمال أي محرر لكود سي شارب سواء كان الفيجوال ستوديو أو غيره فإن بنية الكود تبقى دائما :

```
using System;
class FirstProgram
    {
        static void Main()
        {
                //يكتب الكود هنا
        }
    }
```

ما يهمنا حاليا هو المكان الذي نكتب فيه شفرتنا وهو بين معقوفتي الدالة الرئيسية Main أما مجال الأسماء system و الفئة (الكلاس) FirstProgram سنتطرق إليهما في الفصول القادمة إن شاء الله. سنحاول أن نستغل هذه البنية لنربطها بالمتغيرات لنتقدم خطوة إلى الأمام.



والأن سنقوم بتطبيق ما تعلمناه حول المتغيرات لجعل برنامجنا ينبض بالحياة، و لكن قبل ذلك يجب أن نرصد أهم أنواع البيانات المتواجدة في السي شارب :

1.3 أنواع البيانات :

الفئة (الكلاس)	الإسم المستعمل	الحجم	القيم
System.SByte	sbyte	1	من 128- إلى 127
System.Byte	byte	1	0 إلى 255
System.Int16	short	2	32768- إلى 32767
System.Int32	int	4	214748364 7- إلى 2147483648
System.Uint32	uint	4	0 إلى 4294967295
System.Int64	long	8	9223372036854775808 - إلى 9223372036854775807
System.Single	float	4	$-3,4^{E}+38$ إلى $3,4^{E}+38$
System.Double	double	8	$-1,79^{E}+308$ إلى $1,79^{E}+308$
System.Decimal	decimal	16	$-7,9^{E}+29$ إلى $7,9^{E}+29$
System.Char	char	2	حرف أحادي(UTF-16)
System.Boolean	bool	4	قيمة منطقية (true / false)
System.String	string	*	قيمة نصية

الان بإمكاننا أن ننشئ أول برنامج لنا باستعمال المتغيرات و سيكون كالتالي :

```
using System ;
class FirstProgram
    {
        static void Main()
        {
            int a;
            int b;
            a = 3;
            b = 5;
            Console.Write("La somme de ces deux nombres est  " + (a + b));
            Console.ReadKey();
        }
    }
```

ويمكننا أن نجمع بين إعلان المتغيرات و إعطاء القيم في نفس السطر لتصبح الشفرة :

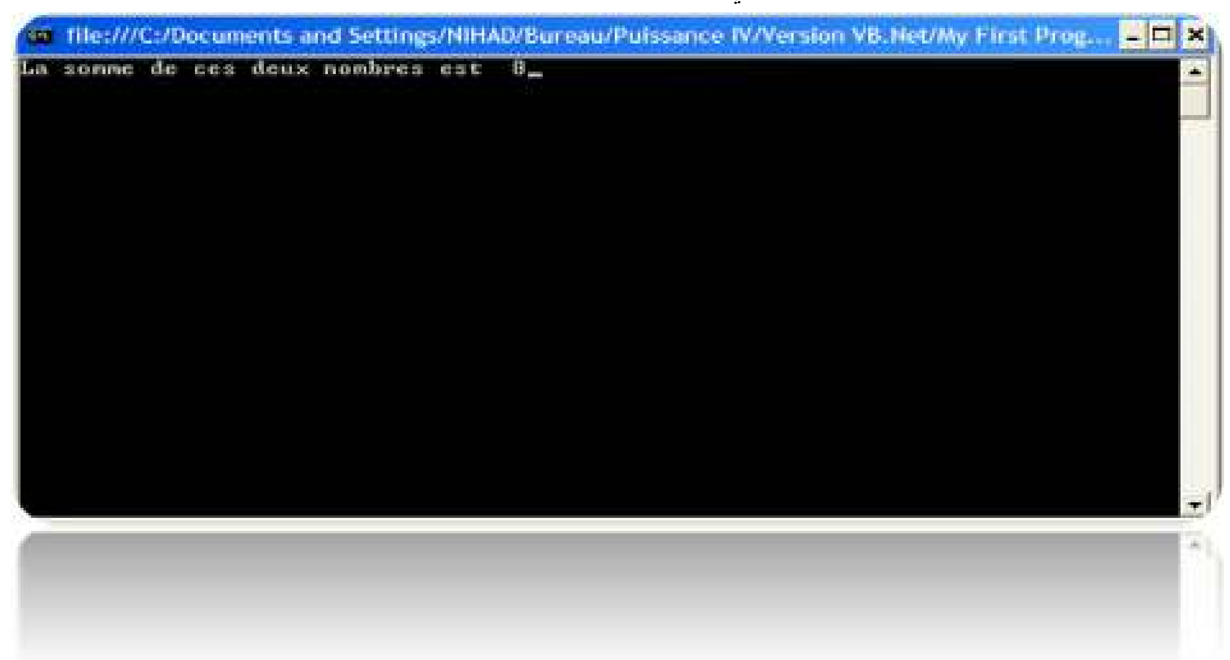



```
using System ;
class FirstProgram
    {
        static void Main()
        {
            int a=3,b=5;
            Console.Write("La somme de ces deux nombres est  " + (a + b));
            Console.ReadKey();

        }
```

سبق و أن شرحنا بنية البرنامج ولكن الجديد هنا هو الدالتين write() و ReadKey() و هما توجدان ضمن الفئة (الكلاس) console، الأول يقوم بطباعة النتيجة على الشاشة و الثاني يقوم بإيقاف الشاشة إلى حين الضغط على أي زر من لوحة المفاتيح. وللإشارة فإنه يجدر التنبيه إلى أن كل سطر مستقل بذاته ينبغي أن ينتهي بفاصلة منقوطة (؛). وكذلك حالة الأحرف فلغة السي شارب حساسة جدا لحالة الأحرف الصغيرة و الكبيرة. وللربط بين نصين علينا استعمال علامة الجمع (+).

وبذلك فبعد التنفيذ ستكون النتيجة كما يلي :



إذا تمكنت من استيعاب هذه الفقرة فقد حققت المبتغى و إن كان العكس حاصلا لا تيأس و ركز معنا في الأمثلة القادمة.



1.4 الثوابت :

مثل المتغيرات لكن قيمتها تبقى ثابتة (جلي ذلك من اسمها !) الإعلان عنها يكون باستعمال الكلمة const:

```
const int a = 5;
```

1.5 الروابط :

حتى يتم التعامل مع المتغيرات و إجراء عمليات عليها وما إلى ذلك فنحن بحاجة إلى رصد هاته الروابط :

1. الروابط الرياضية :

وهي تلك التي تستعمل من أجل القيام بالعمليات الحسابية و نلخصها هنا في هذا الجدول :

الرابط	دوره
+	عملية الجمع
−	عملية الطرح
/	عملية القسمة
*	عملية الضرب
%	باقي القسمة

مثال :

```
using System ;
class Operators
    {
        static void Main()
        {
            int a = 10, b = 6, Somme, Différence, Produit;
            float Division;
            Somme = a + b;
            Différence = a - b;
            Produit = a * b;
            Division = a / b;
            Console.Write("Somme={0}, Dif={1}, Produit={2}, Div={3}",
                           Somme, Différence, Produit, Division);
            Console.ReadKey();
        }
    }
```



أعتقد أن المثال واضح ما عدا استعمال المعقوفات{ } لإعطاء القيم، الشيء الذي نستطيع فعله أيضا بعلامة (+) "راجع الفقرات السابقة"، غير أن هذه الطريقة مستحسنة و بسيطة حاول فقط أن تتأمل طلعتها البهية.

2. الروابط المنطقية :

تقوم بإرجاع صحيح أو خطأ حسب العملية المنطقية و هي كالتالي :

الرابط	دوره
&&	وتعني واو المعية (و) تعيد قيمة صحيحة إذا كانت جميع الأطراف صحيحة.
||	وتعني أو للإختيار تعيد قيمة صحيحة إذا كان على الأقل أحد الأطراف صحيحا.
cond ? var1 : var2	تعيد var1إذا تحقق Condو Var2 إذا تحقق العكس.

3. روابط المقارنة :

وتستعمل من أجل المقارنة بين المتغيرات :

الرابط	دوره
is	للتحقق من نوع المتغير
<	أصغر قطعا
>	أكبر قطعا
<= , >=	أصغر من أو يساوي ، أكبر من أو يساوي
!=	يخالف
==	يساوي

4. روابط إعطاء القيم :

الرابط	دوره
=	لإعطاء قيمة لمتغير
+= ,- =, /=, *=, %=	لها نفس دور العمليات الحسابية العادية
++ , - -	تزيد أو تنقص قيمة المتغير بواحد

لا تجعل الفشل يسيطر عليك من مجرد النظر إلى هاته الروابط، فهي سهلة كشرب الماء (كما نقول نحن المغاربة) فما عليك يا عزيزي سوى التركيز في استعمالها ومحاولة استيعابها.



1.6 أوامر الإدخال و الإخراج

لقد رأينا فيما سبق الدالة write التي تمكننا من طباعة النتائج و إخراجها للمستعمل ، و الآن سوف نتعرف على الدالة التي تمكننا من قراءة القيم المدخلة و التعامل معها و ذلك عن طريق الدالة ()ReadLine،وإليك هذا المثال الذي سنقوم بشرحه و لكن بعد تأملك إياه .

```
using System ;
class ReadValue
    {
        static void Main()
        {
            string Name;
            Console.Write("Enter your name ");
            Name = Console.ReadLine();
            Console.Write("Hello " + Name);
            Console.ReadKey();
        }
    }
```

في اعتقادي يبدو المثال واضحا عن كيفية استعمال دالة القراءة ، سأشير فقط إلى أنها تعيد لنا قيمة نصية، لهذا فعند التعامل مع قيم رقمية يلزمنا أن نحول نوع الدالة وذلك كما يلي :

```
using System ;
class ReadValue
    {
        static void Main()
        {
            int Age;
            Console.Write("Enter your Age ");
            Age = Convert.ToInt32(Console.ReadLine());
            Console.Write("Your age is " + Age);
            Console.ReadKey();
        }
    }
```

آمل أن تتضح لك الرؤية ، لقد قمنا بتحويل القيمة النصية التي تعيدها لنا الدالة ()ReadLine إلى قيمة رقمية عن طريق الدالة ToInt32الموجودة ضمن الفئة (الكلاس) Convert، هذه الفئة تضم العديد من الدوال التحويلية (أنظر الشكل أسفله).

```
class ReadValue
{
    static void Main()
    {
        int Age;
        Console.Write("Enter your Age ");
        Age = Convert.
        Console.Wri   ToDateTime
        Console.Rea   ToDecimal
    }                 ToDouble
}                     ToInt16
                      ToInt32
                      ToInt64
                      ToSByte
                      ToSingle
                      ToString
                      ToUInt16

int Convert.ToInt32(string value, int fromBase) (+ 18 surcha
Convertit la représentation System.String d'un nombre dans u
signé 32 bits équivalent.

Exceptions :
   System.ArgumentException
   System.FormatException
   System.OverflowException
```

1.7 البنية الشرطية :

1. باستعمال if

وهي من أهم الأمور التي يحتاجها المبرمج للتعامل مع المعطيات وللتأكد من صحتها أو خطئها ، وصيغتها كما يلي :

```
if (Condition == true)
            {
                //instruction
            }
```

Condition هي الشرط الذي نريد التحقق منه فإذا كان صحيحا قمنا بالأوامر اللازمة. بالنسبة للكلمة الملونة بالأخضر instruction المسبوقة ب // فتسمى تعليقا، أي لا يتم تنفيذها لأنها ليست من أسطر البرمجة،وإنما مجرد ملاحظة . وتكتب التعاليق في السي شارب بطريقتين :

الطريقة الأولى :



التي شرحناها قبل قليل و هي لكتابة التعليق في سطر واحد، فإن تعدى تعليقنا السطر وجب علينا إضافة // في بداية كل سطر.

مثال :

```
static void Main()
        {
            //This is a comment
            //This is another Comment
        }
```

الطريقة الثانية :

عندما نريد أن نكتب تعليقا يمتد إلى أكثر من سطر، نستعمل هذا الرمز */ في بداية التعليق و ننهيه ب /*
وهذا مثال على استعماله :

```
static void Main()
        {
           /* This is a comment
            This is another Comment */
        }
```

و الآن سنعود إلى بنيتنا الشرطية من أجل إكمال ما بدأنا ، ففي حالة ما إذا أردنا أن نقوم بعمل أمر ما بعد تحقق شرط و القيام بأمر آخر في حالة تحقق العكس وجب علينا استعمال :

```
if (Condition == true)
            {
                //instructions
            }
            else
            {
                //other instructions
            }
```

فإذا كانت العملية تحتاج إلى أكثر من شرط يلزمنا القيام ب :

```
if (Condition == value)
            {
               // instructions          إذا تحقق الشرط الأول
            }
            else if (Condition == OtherValue)
            {
                //other instructions    إذا تحقق الشرط الثاني
            }
            else
            {
                //Other instructions   إذا لم يتحقق أي شرط
            }
```

ويمكننا استعمال else if بقدر عدد الشروط التي نعالج.

وهذا برنامج نستعرض فيه أهم ما تعلمناه إلى حد الآن :

```
using System ;
class Condition
    {
        static void Main()
        {
            int Age;
            Console.WriteLine("How old are you? : ");
            Age=Convert.ToInt32(Console.ReadLine());
            if(Age<=18 && Age>0)  //الروابط المنطقية && تعني (و) راجع
            {
                Console.Write("You are young");
            }
            else if (Age>=18 && Age<140)
            {
                Console.Write("You are adult");
            }else
            {
                Console.Write("Error age !!");
            }
            Console.ReadKey();
        }
    }
```

2. البنية الشرطية باستعمال Switch

وهي لا تعوض if وإنما تفيد في بعض الحالات و صيغتها هي :

```
static void Main()
        {
            switch (Expression) //المتغير الذي سنجري عليه التحقيق
            {
                case 1:        // إذا كانت قيمته هي 1 نقوم بما يلي
                    //instructions;
                    break;
                case 2:        // إذا كانت قيمته هي 2 نقوم بما يلي
                    //other instructions
                    break;
                default:      // الأوامر الإفتراضية في حالة عدم تحقق أي شرط
                    //Defalut instructions
                    break;
             }

        }
```

نقوم بالتحقق من القيمة المدخلة ، فإذا تحققت إحدى الحالات نقوم بالأوامر المرتبطة بها ، وإذا لم تتحقق أي حالة نقوم بأوامر افتراضية.

وحتى نقرب لك المفهوم أكثر تأمل هذا النموذج

```
using System ;
class Switching
    {
        static void Main()
        {
            string Job;
            Console.Write("What's your job ?");
            Job = Console.ReadLine();
            switch (Job)
            {
                case "Doctor" :
                    Console.Write("You are a doctor");
                    break;
                case "Professor" :
                    Console.Write("You are a Professor");
                    break;
                default:
                    Console.Write("Job unKnown !!");
                    break;

             }
            Console.ReadKey();
        }
```

كلمة **break** تمكننا من الخروج من الشرط فورا بعد تحققه .

3. البنية الشرطية عن طريق الرابط الثلاثي :

للقيام بعملية التحقق بتزامن مع عملية إعطاء القيم نحتاج إلى استعمال هذه البنية التي سبق أن رأيناها في فصل الروابط، و صيغتها كما يلي :

```
Expression ? Valeur1 :valeur2
```

Expression هي العبارة التي ستتحقق منها فإن كانت تعيد لنا القيمة true سيتم تنفيذ valeur1 و إن كان العكس سيتم تنفيذ valeur2،و هذا مثال على كيفية استعماله :




```
class Ternaire
    {
        static void Main()
        {
        string Name;
        string Expression;
        Console.WriteLine("What's your Name?");
        Name = Console.ReadLine()
        Expression = Name == "khalid"?"It's my name!":"Nice to meet you !!" + Name;
        Console.WriteLine(Expression);
        Console.ReadKey();
        }}
```

سأعلق بإيجاز على هذا المثال، فالعبارة المسطرة هي التي تعنينا ، وهي تقول إذا كانت قيمة المتغير Name تساوي khalid نظهر الرسالة الأولى it's my name وإن كان العكس نظهر الرسالة الثانية Nice to meet you مرتبطة بقيمة المتغير.

1.8 البنية التكرارية:

لا غرو أن تكرار الأوامر له نفع كبير على المبرمجين فلولا هذه الإمكانية لكان من الصعب إنتاج البرامج بكفاءات عالية، لهذا ركز معي جيدا في هذه الفقرة و حاول أن تستأنس بكل مثال نسرده لك، حتى نحقق المبتغى و ننهض عن المائدة ونحن راضون (لا تستغرب من تعابيري هذه فأنا شاعر أكل الدهر عليه و شرب).

للقيام بتكرار البيانات تتيح لنا لغة السي شارب هذه الطرق :

1. الطريقة الأولى :

باستعمال For

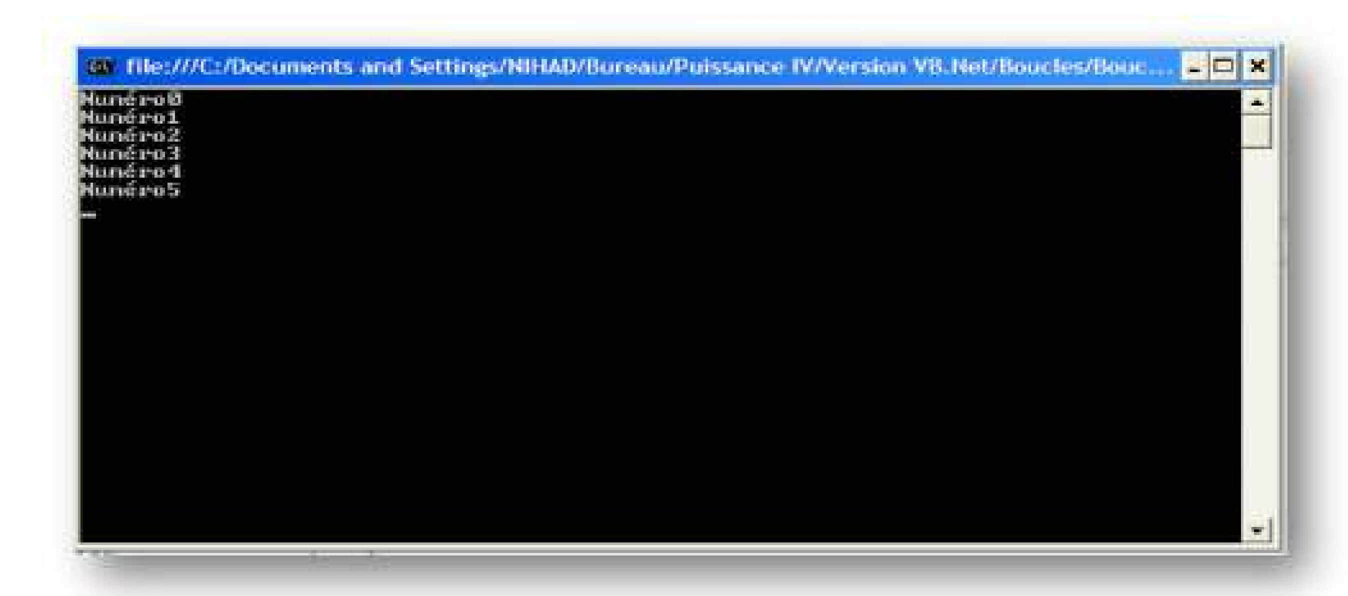
وصيغتها كما يلي :

```
for (i = 0; i < 6; i++)
            {
                Console.WriteLine("Numéro" + i);
            }
```

أعطينا في الأول قيمة ابتدائية للمتغير i و هي 0 ثم بعد الفاصلة المنقوطة حددنا عدد المرات التي نرغب فيها بتكرار الأوامر و هي 6 و أخيرا حددنا مقدار الزيادة و هنا ستزيد قيمة المتغير i ب 1 ، بمعنى آخر ستكون قيمته في الأول 0 و في المرة الثانية ستزداد لتصبح واحد و هكذا دواليك حتى نستوفي الست مرات.

و في مثالنا هذا ستكون النتيجة كما يلي :





ودونك هذا المثال لتدعيم الفهم:

```
using System ;
class Boucles
    {
        //Programme pour calculer le factoriel
        static void Main()
        {
            int i,Nombre;
            int Factoriel = 1;
            Console.WriteLine("Donner un nombre");
            Nombre = Convert.ToInt32(Console.ReadLine());
            for (i = Nombre; i > 0; i--)
            {
                Factoriel = Factoriel *i   ;
             }
            Console.WriteLine("Le factoriel de ce nombre est " +
                   Factoriel);
            Console.ReadKey();
        }
```

قمنا بالإعلان عن متغيرين رقميين i و Nombre، و متغير ثالث Factoriel أعطيناه 1 كقيمة بدئية لسبب سنعرفه بعد قليل ، وطلبنا من المستعمل أن يدخل لنا رقما لنحسب دالة العاملي له ، وبعد ذلك قمنا بعمل حساب العاملي وذلك ببدء المتغير i من القيمة التي يدخلها المستعمل إلى صفر و جعلنا التغير تناقصيا بواحد، ثم أخذنا المتغير Factoriel الذي كنا قد أعطيناه 1 كقيمة بدئية و أضفنا إليه جداءه مع المتغير i ، بمعنى آخر لو أن المستعمل أدخل لنا القيمة 3 فإن التكرار سيكون من 3 إلى 0 تناقصيا و في كل مرة نحافظ على قيمة Factoriel القديمة و نضيف إليها جداءها مع قيمة المتغير i.



الآن فهمت لماذا أعطينا المتغيرFactoriel القيمة 1 كقيمة بدئية لأننا لو أعطيناه 0 لكانت النتيجة دائما 0 لأننا نستعمل الجداء.

أعتقد أن الأمر معقد شيئا ما و لكن نفعه در، حاول فقط أن تراجع المثال بهدوء و تركيز و سوف تفهم كنه هذه البنية، لأنها تبدو صعبة في الأول وهي عكس ذلك سهلة كشرب الماء (لقد أرهقتك بهذه العبارة و لكن لاعليك ستنسى ذلك و أنت تتأمل المثال ، حظا موفقا).

2. الطريقة الثانية :

باستعمال While

لها نفس دور البنية السابقة غير أنها تخالفها في الصيغة :

```
while (Condition)     // الشرط الذي بتحققه نكرر الأوامر
        {
            //instructions الأوامر التي نقوم بتكرارها
        }
```

إذا ترجمنا الصيغة فمعناها "ما دام الشرط متحققا سنكرر الأوامر المطلوبة".

وهذا المثال سنضرب فيه عصفورين بحجر واحد سوف نتعرف على كيفية استعمال while زيادة على الأمر goto (تعني اذهب إلى)

مثال :

```
using System ;
class Boucles
    {
        static void Main()
        {
            int Valeur;
            Point:
            Console.WriteLine("Devinez la valeur ");
            Valeur = Convert.ToInt32(Console.ReadLine());
            while (Valeur != 5)
            {
                Console.WriteLine("La valeur est fausse réessayer ");
                goto Point;
            }
            Console.WriteLine("Vous avez gagné !!");
            Console.ReadKey();
        }
    }
```

طلبنا من المستعمل أن يدخل قيمة ثم تحققنا من أنها تخالف 5 باستعمال while فما دام المستعمل لم يدخل لنا القيمة 5 سوف نعلمه أن الاحتمال خاطئ ثم نعيد الكود إلى نقطة القراءة التي أعطيناها الاسم Point بواسطة الأمر goto، وعند عدم تحقق الشرط valeur تخالف 5 سوف نخرج عن التكرار ونظهر رسالة الفوز للمستعمل لأنه قام بإدخال الرقم 5.

أظن أن الأمر واضح الآن، بإمكانك أن تنشئ مجموعة من الأمثلة بنفسك و تجربها فالأمر بسيط جدا ، ولا تنسى التكرار باستعمال for فهو الأكثر استعمالا فاحرص أن تتمكن من استيعاب البنية التكرارية لأن ما رأيناه في هذا الفصل شامل بقي لنا فقط أن نرى آخر بنية تكرارية وهي تشبه إلى حد بعيد while .

3. الطريقة الثالثة :

باستعمال Do

نقوم في الأول بالأوامر ثم نتحقق في الأخير من تحقق الشرط إنه باختصار while بصيغة أخرى تتمثل في :

```
do
{
    //Instruction          الأوامر التي نرغب في تكرارها
} while (Condition);      // الشرط الذي يتحققه نعيد الأوامر
```

وهذا مثال على استعمالها :

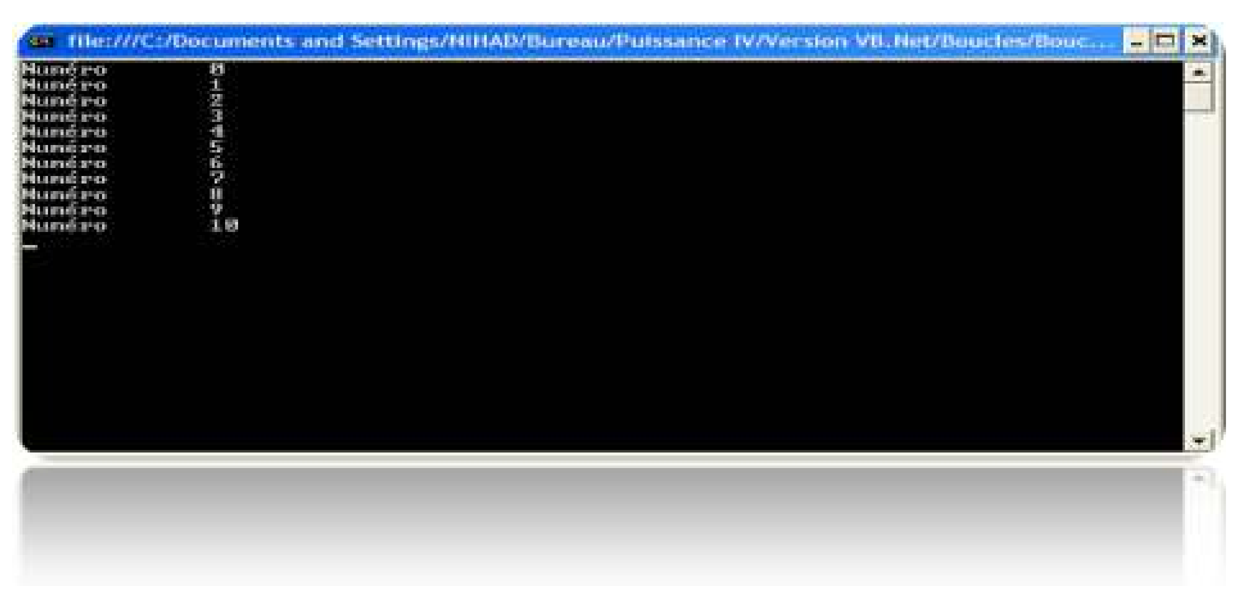
```
class Boucles
    {
        static void Main()
        {
            int Number=0;
            do
            {
                Console.WriteLine("Numéro       "+ Number++);
            } while (Number>0 && Number<=10);
            Console.ReadKey();
        }
    }
```

قمنا بإظهار الأرقام من 1 إلى 10 ، وذلك كما يلي :

ما دام المتغير Number أكبر من 0 و أصغر من أو يساوي 10 سنظهر قيمته زائد 1.

والنتيجة كما يلي :





4. Foreachالتكرار باستعمال

سنراه لاحقا في فصل المصفوفات

5. الخروج من البنية التكرارية :

نحتاج أحيانا إلى الخروج من أوامر التكرار و لفعل ذلك نستعمل الأمر break ، و للقيام بالعكس أي بمتابعة التكرار بعد تحقق شرط ما نستعمل الأمر continue و لكي تفهمهما جيدا تأمل هذا المثال و اكتبه و نفذه لترى النتيجة .

```
using System ;
class Boucles
    {
        static void Main()
        {
            int i;
            for (i = 0; i < 6; i++) {
                Console.Write(i);
                if (i< 3) continue;
                break;
        }
            Console.ReadKey();
    }
```

1.9 رموز الإختصار Les charactéres d'échappement(Escape characters)

وهي رموز تستعمل لتسهيل عملية الكتابة فمثلا العلامة ' لا نستطيع طباعتها على الشاشة لأن البرنامج سيظهر لنا خطأ و سيمنعنا من التنفيذ ، كما أنه لا يليق بالمبرمج الذي لا يرضى بأصابع الإزدراء أن تشير إليه أن يقوم في كل مرة باستعمال console.writeLine() من أجل طباعة سطر جديد فلإظهار هذا

النموذج :

```
file:///C:/Documents and Settings/NIHAD/Bureau/Puissance IV/Version VB.Net/Escape chara...
                **Menu**

                01-Csharp
                02-VB
                03-Java
```

سنحتاج إلى كتابة كل هذه الشفرة:

```
using System ;
class Program
    {
        static void Main()
        {
            Console.WriteLine("                 **Menu**");
            Console.WriteLine("                 01-Csharp");
            Console.WriteLine("                 02-VB");
            Console.WriteLine("                 03-Java");
            Console.ReadKey();
        }
    }
```

أما الآن وباستعمال رموز الإختصارات فقد أصبح الأمر ممكنا في جملة واحدة:

```
class Program
    {
        static void Main()
        {
            Console.WriteLine("\t\t**Menu**\n\n\t\t01-Csharp\n\t\t02-
VB\n\t\t03-Java");
            Console.ReadKey();
        }
    }
```



ستصل إلى نفس النتيجة، لكن لا تتشاءم من منظرها فذلك لا يليق بك أنت يا من عودتنا على احترام هيئة الأكواد مهما كانت أشكالها .

العديد من لغات البرمجة و سكريبتات الويب تستعمل هذه الرموز نظرا لمردوديتها و فاعليتها ولعل أبرز مثال لغة الجافا و السي بلس بلس و الجافا سكريبت، كما أنك مرغم يا عزيزي لا بطل على استعمالها في حال أردت طباعة كلمة بها علامة ' أو " أو غيرها وهذا الجدول يعرض أبرز الرموز و ليس كلها:

الرمز	دوره
\'	لكتابة الرمز '
\"	لكتابة الرمز "
\\	لكتابة الرمز \
\a	لإصدار صوت تحذير Beep
\n	سطر جديد
\t	مسافات فارغة Tabulation
\b	لمسح الحرف الأخير من الكلمة



مجموعات البيانات



2. مجموعات البيانات

1. المصفوفات Les tableaux (Arrays)

وهي تشمل مجموعة من المتغيرات التي لها نفس النوع ويكون الإعلان عنه كما يلي :

```
Type[] Array=new Type[N];
```

حيث Type هو نوع المصفوفة و Array اسمها و N عدد عناصرها.

وهذا مثال على استعمالها :

```
using System ;
class Tableau
    {
        static void Main()
        {
         int[] Array = new int[5];//قمنا بالإعلان عن مصفوفة بها 5 عناصر

         int i,j,Temp;

         for (i = 0; i < Array.Length; i++)//انطلقنا من 0 إلى نهاية المصفوفة
            {
                Console.WriteLine("Donner l\'element " + i);

                            //  بدأنا بقراءة العناصر المدخلة

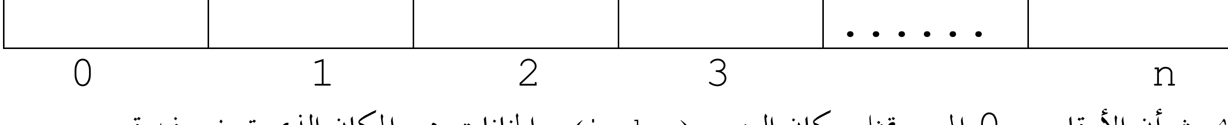
                  Array[i] = int.Parse(Console.ReadLine());

                                        //  convert.ToInt32 لها نفس دور
            }


        for (i = 0; i < Array.Length; i++)
//التكرار الأول ينطلق من أول عنصر بالمصفوفة إلى آخر عنصر//
                for (j = i + 1; j < Array.Length; j++)
//التكرار الثاني ينطلق من ثان عنصر بالمصفوفة إلى آخر عنصر//
                {
```

```
                if (Array[i] > Array[j]) //نقارن بين عنصرين متتابعين
// إذا كان العنصر الأول أكبر من العنصر الثاني
                    {
                    Temp = Array[i];/*نخزن قيمة العنصر الأول في المتغير
                                         Temp */
                    Array[i] = Array[j]; //و قيمة العنصر الثاني في مكان
العنصر الأول
                    Array[j] = Temp; //ثم في الأخير نخزن قيمة المتغير في مكان
العنصر الثاني
                    }
                }
            {
            }
            for (i = 0; i < Array.Length; i++)
//نقوم بهذا التكرار من أجل إظهار جميع قيم المصفوفة بالترتيب الجديد
            {
                Console.WriteLine("Le tri de ces nombres par ordre
croissant est : " + Array[i]);
             }
            Console.ReadKey();
        }
```

هذا البرنامج يقوم بترتيب عناصر المصفوفة من الأصغر نحو الأكبر ، لا يغرنك شكله فهو سهل حاول فقط أن تركز جيدا .فمن خلاله ستتمكن من معرفة أهمية المصفوفات و كيفية استعمالها ، فالمصفوفات بمثابة جدول متكون من أعمدة و أسطر ، في مثالنا هذا اشتغلنا على مصفوفة أحادية الأبعاد ،بمعنى أنها تتكون من بعد واحد أي أن تمثيلها سيكون كما في الشكل التالي :



بحيث أن الأرقام من 0 إلى n تمثل مكان العنصر (index) و الخانات هي المكان الذي توضع فيه قيم العناصر وهذا المثال المرفق سيوضح الفكرة إن شاء الله.

```
string[] Names=new string[4];
Names[0] = "Abu bakr";
Names[1] = "Omar";
Names[2] = "Ali";
Names[3] = "Othman";
```

هذه مصفوفة متكونة من أربعة عناصر ، العنصر الأول أي الموجود في المكان 0 قيمته "Abu bakr" ينبغي عدم الخلط بين مكان العنصر و قيمته فالمكان هو موضعه في المصفوفة أما القيمة فهي العبارة التي نخزنها في الموضع.بالنسبة لتمثيل هذه المصفوفة سيكون كما يلي :

Abu bakr	Omar	Ali	Othman
0	1	2	3

استعمال foreach

وهي وسيلة أخرى من وسائل تكرار الأوامر إلا أن مايميزها هو أنها سهلة وتعمل على الاشتغال على كل عنصر في مجموعة على حدى، فيما يخص المجموعات التي تعرفنا عليها نأخذ المصفوفات كنموذج ، صيغة هذه البنية كالآتي :

```
foreach (TypeVariable variable in Collection)
            {
                //Instructions
            }
```

ويجب أن يكون نوع المتغير مثل نوع المجموعة،كمثال عل استعمال هذه البنية نعرض ما يلي :

```
using System ;
class Program
    {
        static void Main()
        {
            string[] Name = new string[2];
            Name[0] = "mohamed";
            Name[1] = "khalid";
            foreach (string Nom in Name)
            {
                Console.WriteLine(Nom.ToUpper());
            }
            Console.ReadKey();
        }
    }
```

قمنا بتحويل أحرف أي عنصر في المصفوفة من حالتها الصغيرة إلى حالتها الكبيرة.

ستكون النتيجة كما يلي :

```
MOHAMED
KHALID
```

وتوجد كذلك مصفوفات متعددة الأبعاد بمعنى متكونة من أسطر و أعمدة و المثال التالي يعرض كيفية الإعلان عن مصفوفة ثنائية الأبعاد :

```
int[,] Array = new int[3, 4];
```

هذه مصفوفة ثنائية الأبعاد متكونة من ثلاثة أسطر و أربعة أعمدة، وتمثيلها كما يلي :

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

وهذا مثال على استعمالها :

```
using System ;
class Matrice
    {
        static void Main()
        {
            int[,] Matrice = new int[3, 4];
            int i, j;
            for(i=0;i<=2;i++)
            {
                for (j = 0; j <= 3; j++)
                {
                    Console.WriteLine("Donner l\'élément :(" + i + "," + j
+ ")");

                    Matrice[i, j] = int.Parse(Console.ReadLine());
                }
            }
            for (i = 0; i <= 2; i++)
            {
                for (j = 0; j <= 3; j++)
                {
                    Console.WriteLine("La valeur de l\'élément :(" + i +
"," + j + ") est " + Matrice[i,j] );
                }
            }
            Console.ReadKey();
        }
```

في هذا المثال لم نقم سوى بالإعلان عن مصفوفة ثنائية الأبعاد مكونة من ثلاثة أسطر و أربعة أعمدة ثم بعد ذلك أدخلنا قيم كل عنصر من المصفوفة عن طريق بنية تكرارية مزدوجة للمرور على كل خلايا المصفوفة ثم في الأخير قمنا بطباعة القيم المدخلة بنفس الطريقة، الغاية من المثال جعلك قادرا على التعامل مع المصفوفات من خلال البنية التكرارية، ولك أن تعيد المثال بأسلوبك أو أن تحاول تعديله و تطويره كأن تجعله يقوم بترتيب عناصر المصفوفة كما عملنا سابقا (ستحتاج نفس الطريقة وإن كان لديك البديل فذلك أفضل على شرط أن تكون قد تمكنت من استيعابه).

يمكن للمصفوفة أن تتعدى بعدين، أي أن تصبح ثلاثية أو رباعية أو أكثر، تبعا لحاجياتك البرمجية، سأشير فقط إلى أن الطريقة للتعامل مع المصفوفات الغير الأحادية يقتضي منك فصل كل بعد بفاصلة عن البعد الآخر، فمثلا إن أردت التعامل مع مصفوفة ثلاثية الأبعاد ستحتاج إلى فاصلة أخرى وهذا مثال على ذلك :

```
char[,,] TriDimensions = new char[2, 3, 5];
```



الأمر في غاية البساطة قم فقط بترع نظاراتك السوداء .

2. اللوائح Les listes(List)

لعلك لاحظت أننا مرغمون على تحديد عدد عناصر المصفوفة عند الإعلان عنها، لا تقلق فاللوائح أتت لتحل لنا هذا المشكل و غيره ،حيث نستطيع تخزين كل الأنواع بل حتى الفئات التي سنراها قادما إن شاء الله ، ويكون الإعلان عن اللائحة بالطريقة التالية:

```
List<Type> nomDeLaListe = new List<Type>();
```

حيث Type هو نوع العناصر و nomDeLaListe اسم اللائحة.

و نستطيع إضافة العناصر إلى اللائحة عن طريق الدالة Add :

```
List<String> Names = new List<string>();
Names.Add("Mohamed");
Names.Add("Khalid");
```

أو إضافتهم عند البداية عن طريق:

```
List<String> Names = new List<string> { "Mohamed", "Khalid" };
```

أما إظهار القيم و التعامل مع عناصر اللائحة فهو نفسه مع المصفوفات ، سنستعمل هذا المثال للاستئناس فقط foreach .

```
using System;
class Program
{
    static void Main(string[] args)
    {
        List<String> Names = new List<string> { "Mohamed", "Khalid" };
        foreach (String Name in Names)
        {
            Console.WriteLine(Name);
        }
        Console.ReadKey();
    }
}
```

تتوفر اللائحة على العديد من الدوال و الوظائف التي تسهل علينا العديد من الأشغال مثل clear التي تمسح عناصر اللائحة و count التي تعود لنا بعدد العناصر وغيرها كثير سنترك معها لتتعرف عليها.

3. المعددات Enumerations(Enum)

المعددات عبارة على نوع من البيانات تضم عناصر مرتبة حسب مجال رقمي يبدأمن 0 وينتهي بموضع آخر عنصر ، وسنأخذ كمثال على سبيل الإيضاح درجات سهولة لعبة معينة وغالبا ما تكون هذه الدرجات مصنفة كالتالي(سهل،متوسط،صعب)، في هذه الحالة يمكننا أن نصنع نوعنا الخاص من البيانات لدرجات اللعبة باستعمال المعددات وسيكون ذلك كالآتي :

```
enum Niveau
      {
          Facile,
          Moyen,
          Difficile
      };
```

Niveau هو إسم المعددة و العناصر الثلاثة هي القيم الممكن التعامل معه عند استخدام المعددة كنوع من البيانات، أما فيما يخص الترتيب فالعنصر الأول يحتل المكان 0 وهكذا دواليك إلى آخر عنصر في المعددة.

للإعلان عن متغير من هذا النوع نعتمد نفس الطريقة المعروفة:

```
Niveau NiveauJeu;
```

وهنا مثال بسيط على استعمال هذه المعددة :

```
class Enumerations
    {
        enum Niveau{Facile,Moyen,Difficile};
        static void Main()
        {
            Niveau NiveauJeu = Niveau.Facile;
            Console.WriteLine("*************Niveau du jeu**************");
            if (NiveauJeu == Niveau.Facile)
            {
                Console.WriteLine("Ce niveau est facile ");
            }

        }
      }
```

لا أظن أن الأمر يضم لبسا ، المهم أن تكون قد أدركت الغاية من المعددات التي سنلخصها في هذه الجملة وهي جعل المبرمج قادرا على ابتكار نوع من البيانات خاص بالوضعية التي يتعامل معها إلا أن عطاء المعددات يبقى محدودا أمام التراكيب التي سنراها بعد هذه الفقرة.

4. التراكيب Structure (Struct)

تعتبر مدخلا مهما للبرمجة الشيئية حيث تشبه إلى حد بعيد بنية الفئات(الكلاس)، كونها تضم متغيرات و دوال و خصائص و مشيدات كما هو الحال مع الفئات . لا ترتبك فأنا أعتقد أن هذه المصطلحات تبدو جديدة بالنسبة إليك لكنها ليست صعبة كما تتصور فهي سهلة كشرب ...

أخي،ركز في هذا المثال جيدا وبعدها سنمضي قدما لاستيعاب مفهوم التراكيب :

```
        struct Personne
        {
            string Nom;
            string Adresse;
            short Age;
        }
```

هذا التركيب الذي سميناه Personne يضم بين دفتيه ثلاثة متغيرات تعتبر مرتبطة به، فليس هناك شخص بدون اسم أو عنوان أو سن ، لهذا فكل كائن مستقل بذاته و له ما يميزه عن الكائنات الأخرى من صفات و

مهام سنعتبره تركيبا، فعلى سبيل المثال يمكننا أن نعتبر السيارة تركيبا لأنها كائن مستقل له خاصياته كالإسم و النوع والرقم واسم المصنع..

هكذا يمكننا الإعلان عن متغير من نوع التركيب ، له نفس المتغيرات الجزئية، تماما كما يلي :

```
class Structure
    {
        struct Personne
        {
            public string Nom;
            public string Adresse;
            public short Age;
        }

        static void Main()
        {
            Personne Person;
            Console.WriteLine("Donner votre nom");
            Person.Nom = Console.ReadLine();
            Console.WriteLine("Donner votre Adresse");
            Person.Adresse = Console.ReadLine();
            Console.WriteLine("Donner votre Age");
            Person.Age =Convert.ToInt16(Console.ReadLine());
           //Affichage
          Console.WriteLine("Votre nom est:{0} , votre adresse est: {1} " +
                            " , votre Age est: {2}" , Person.Nom,
                            Person.Adresse, Person.Age);
            Console.ReadKey();

        }
    }
```

قم بتجريب البرنامج لفهم مضمونه فهو لا يقوم سوى بالإعلان عن تركيب اسمه Personne له متغيراته الخاصة Nom وAdresse وAge ثم بعد ذلك نقوم بالإعلان عن متغير جديد من نوع التركيب و فيه نقوم بخزن القيم المدخلة من أجل إظهارها ، أعتقد أنك فهمت الآن أهمية استعمال التراكيب ، ولكن عليك أن تدعم فهمك بأمثلة من صنعك ففهم هذه الفقرة كليا سيجعلك ترى البرمجة الشيئية بمنظور صائب و بعين مدركة.

5. الدوال Les fonctions (functions)

(وإن كانت ليست ضمن مجموعات البيانات ولكنني ارتأيت وضعها هنا كمدخل للبرمجة الشيئية)

صدقني إن قلت لك لا يخلو أي برنامج من مفهوم الدوال فهي قوامه التي لا يستقيم إلا بها ، ستقول لي في الأمثلة السابقة لم نستعمل الدوال و نجحت الأكواد، ولكنني سأقول لك بل استعملناها و بكثرة ولعل أبرز دليل على صدق كلامي أن كل الأكواد لم تخل من الدالة الرئيسية Main() .

لعل الدالة Main() قد أوحت لك بشيء الآن وهي بنية الدوال على العموم، لكن لاتتسرع سننطلق من البداية و كأن الدالة Main() لا تمت لنا بصلة بمعنى آخر سنجحد فضلها ونتركها طريحة الفراش (هه أمزح فقط أعوذ بالله أن أكون من الجاحدين ! !).

بالنسبة للدوال فهي نوعان : دوال تقوم بمجموعة من المهام لكنها لا تعيد لنا أي قيمة و دوال أخرى تقوم بإرجاع قيمة .

وطريقة الإعلان عن الدوال تكون كما يلي :

```
TypeDeFonction NomDeFonctions(Parametres)
        {
            //instructions
            return valeur;
        }
```

نقوم بتحديد نوع الدالة وإعطائها اسما، ونقوم بفتح الأقواس للإعلان عن المتغيرات التي تقوم الدالة بمعالجتها، فمثلا سنقوم بكتابة دالة تقوم بجمع عددين وتعيد إلينا الناتج :

```
Static int Somme(int N1, int N2)
        {
            return N1 + N2;
        }
```

يجب أن تكون القيمة المعادة المسبوقة بالأمر return من نفس نوع الدالة ، و لاستعمال هذه الدالة سنكتفي فقط بذكر اسمها مع إعطاء قيم المتغيرات N1 و N2.

```
Somme(3, 5);
```

بإمكاننا أن نعطي لمتغير من نفس نوع الدالة قيمتها مثل :




```
int Som = Somme(12, 55);
```

ثم نطبع قيمته على الشاشة أو نقوم بذلك مباشرة دون الحاجة إلى الإعلان عن المتغير :

```
Console.WriteLine(Somme(3, 8));
```

وهذا مثال بسيط يستعرض ما رأينا للتو :

```
using System ;
class Somme
    {
        //La fonction
        static int Somm(int N1, int N2)
        {
            return N1 + N2;
        }
        static void Main()
        {
            int N1, N2;
            Console.WriteLine("Donner le premier nombre :");
            N1 = int.Parse(Console.ReadLine());
            Console.WriteLine("Donner le deuxiéme nombre :");
            N2 = int.Parse(Console.ReadLine());
            Console.WriteLine("La somme de ces deux nombres est :" +
                                        Somm(N1, N2));
            Console.ReadKey();
        }
    }
```

الغاية من استعمال الدوال هي جعل الكود سهل القراءة و كذلك لتفادي تكرار القيام بنفس الأوامر ، فتخيل معي برنامجا يقوم بمجموعة من العمليات الروتينية كيف سيكون شكله أو بالأحرى شكل المبرمج من جراء الملل والتعب،إن استعمال الدوال طريقة ذكية لتعميم الأوامر وتلخيصها في سطر واحد يستقبل القيم إن كانت هنالك قيم.

بالنسبة للدوال التي لا تعيد قيما فهي تعمل بنفس الطريقة باستثناء إقصاءها للأمر return و اعتمادها على الكلمة المفتاح void التي تعني أن الدالة لن تعيد شيئا وكمثال على ذلك انظر إلى الدالة الرئيسية Main().

بعد ذلك تأمل هذا المثال وحاول تطويره.

```
using System ;
class WithoutReturn
    {
        static void MyFunction()
        {
            string Nom;
            Console.WriteLine("Entrer votre nom :");
            Nom = Console.ReadLine();
            Console.WriteLine("Bonjour " + Nom);
            Console.ReadKey();
        }

        static void Main(string[] args)
        {
            MyFunction();
        }
    }
```

البرمجة كائنية التوجه

3. البرمجة الكائنية التوجه

لا تقل لي بأنك نسيت مفهوم التراكيب ، إن كان الأمر كذلك فعد بسرعة إليها و تأملها برهة ثم عد لتكملة هذا الفصل وحاول أن تركز جيدا فهذا النوع من البرمجة هو الذي يحدد مقدار احترافية المبرمجين،ومدى قدرتهم على إبداع كائناتهم البرمجية، خصوصا وأن كل اللغات الذائعة الصيت تدعم البرمجة الشيئية ولعل أشهر هذه اللغات في هذا المجال لغة الجافا ولغتنا هذه، فسمعتها طيبة فيما يخص هذا النوع من البرمجة .

إن أبرز مصطلح في البرمجة الشيئية هو مصطلح الفئات classes ، وتعتبر هذه الأخيرة بمثابة التمثيل الشامل للكائن حيث تضم خصائصه و أفعاله و كل ما يتعلق به منذ بدايته إلى نهايته،و هي قريبة جدا لمفهوم التراكيب بل وتتفوق عليه حيث أن التراكيب تظل محدودة العطاء إذا ما تأملنا الجانب البرمجي بحيث أن التراكيب لا تقبل عملية الوراثة و التي سنتعرف عليها فيما بعد ، وتتفوق الفئات على التراكيب أيضا في بعض الأمور التي سنراها قريبا إن شاء الله، و يكمن الفرق الرئيسي بين هاتين البنيتين في كون المتغيرات التي نعلنها من نوع التراكيب تضع في الذاكرة صورة للتركيب الأصلي بمعنى آخر تأخذ قيمته، أما إذا قمنا بالإعلان عن متغير من نوع فئة معينة فالأمر مختلف تماما حيث لا يتم تكوين نسخة مشابهة للفئة في الذاكرة وإنما يتم فقط التأشير إلى مكانها باعتبارها مرجعا.

ولتكن Arkan تركيبا و فئة للمقارنة بين المفهومين و هذه شفرة كل منهما :

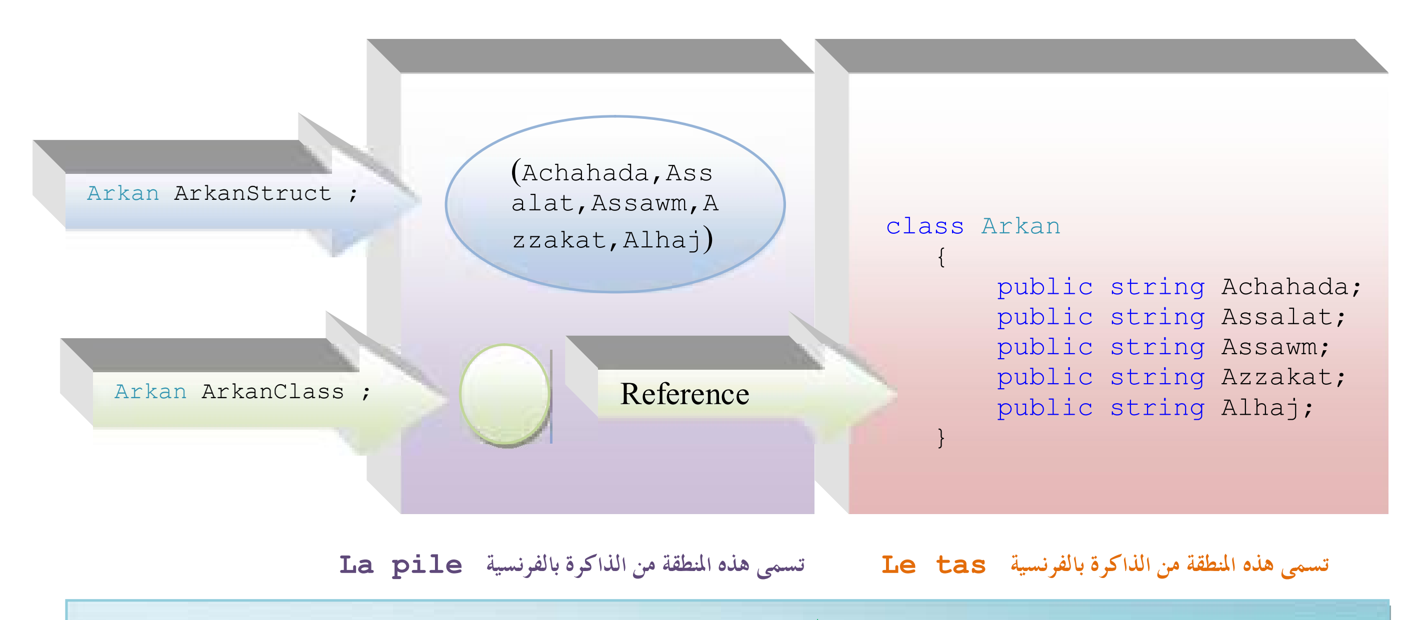
```
class Arkan
    {
        public string Achahada;
        public string Assalat;
        public string Assawm;
        public string Azzakat;
        public string Alhaj;
    }
```

```
Arkan ArkanClass;
```

```
struct Arkan
        {
            public string Achahada;
            public string Assalat;
            public string Assawm;
            public string Azzakat;
            public string Alhaj;
        }
```

```
Arkan ArkanStruct;
```

ولتجسيد العبارة سأدعك تتأمل هذا الشكل الذي يوضح الفرق بين القيمة (value) و المرجع (reference):



المفتاح

القيمة value

المرجع reference

إذن فالفئات نوع بالمرجع و التراكيب نوع بالقيمة ، و المصفوفات التي رأيناها فيما قبل نوع بالمرجع أيضا و هذا الجدول يضم الأنواع الممكنة و طبيعتها أهي بالنوع أم بالمرجع :

الأنواع بالمرجع	الأنواع بالقيمة
الفئات class	جميع الأنواع الرقمية int
المصفوفات Array	جميع الحروف char
النصوص string	الأنواع المنطقية boolean
التفويض delegate	التراكيب struct
	المعددات enum

الدوال التي تنتظر قيما تستعمل أيضا تمرير المتغيرات إما بالمرجع أو بالقيمة وهذا شرح مفصل على كيفية عمل ذلك:

رأينا في السابق كيف تستقبل الدوال قيمها و لم نركز على ذلك ،والآن حان الوقت لعمل ذلك ، فلتمرير المتغيرات إلى دالة معينة نقف أمام أمرين مختلفين تماما :

التمرير بالقيمة :

وهو الأسلوب الإفتراضي حيث تتغير قيمة المتغير في الدالة فقط وتبقى قيمته الأصلية ثابتة لا يطالها التغيير مثلا

```
using System ;
class FunctionByValue
    {
        static void Function(int Number)
        {
            Number = 3;
            Console.WriteLine("La valeur du variable maintenant est :" +
                                   Number);
        }
        static void Main()
        {
            int Num = 10;
            Function(Num);
            Console.WriteLine("La valeur d\'origine est :" + Num);
            Console.ReadKey();
        }
}
```

تأخذ الدالة function المتغير Num و تعطيه القيمة 3 لكنها لا تفقده قيمته الأصلية ، أي أننا إذا أردنا استعمال المتغير Num فإن قيمته الأصلية هي 10 دائما، النتيجة ستكون هكذا :

```
La valeur du variable maintenant est :3
La valeur d'origine est :10
```

التمرير بالمرجع :

ويكون باستعمال المصطلح **ref** وهنا يتم تعويض قيمة المتغير الخارجي بقيمة المتغير الداخلي للدالة مما سيفقده قيمته الأصلية ، لأن التمرير لم يكن بالقيمة وإنما بالمرجع :

وهذا المثال يستعرض ذلك :

```
using System ;
  class FunctionByReference
    {
        static void Function(ref int Number)
        {
            Number = 3 ;
            Console.WriteLine(« La valeur du variable maintenant est : » +
                              Number) ;
        }
        static void Main()
        {
            int Num = 10 ;
            Function(ref Num) ;
            Console.WriteLine(« La valeur d\'origine est : » + Num) ;
            Console.ReadKey() ;
        }
    }
```

هنا ستكون النتيجة مغايرة لأن التمرير كان بالمرجع وليس بالقيمة ، بالتالي ستكون النتيجة هكذا :

```
La valeur du variable maintenant est :3
La valeur d'origine est :3
```

لكن استعمال **ref** يستلزم من المتغير الخارجي Num أن تكون له قيمة بدئية فإن كان هذا المتغير معلنا عنه بدون قيمة سيحدث خطأ عند التنفيذ مضمونه أن المتغير تم استعماله من دون الأشارة إلى قيمته، ولتفادي هذا الخطأ سنحتاج إلى استعمال الكلمة المفتاح **out** والتي تعلم المترجم أن المتغير Num هو متغير للخروج فقط أي لا نستعمله إلا من أجل تعويض المتغير الداخلي للدالة. لهذا ستكون قيمته الأصلية هي نفسها قيمة المتغير الداخلي.

ويكون استعمال الكلمة **out** كما يلي :

```
using System;
   class Program
   {
       static void Function(out int Number)
       {
           Number = 3 ;
           Console.WriteLine(" La valeur du variable maintenant est : "
+Number) ;
       }
       static void Main()
       {
           int Num ;
           Function(out Num) ;
           Console.WriteLine(" La valeur d\'origine est : " + Num) ;
           Console.ReadKey() ;
       }

   }
```

يبدو أن شكل الفئات أصبح الآن مألوفا نسبيا، سنقوم فقط بذكر البنية العامة للفئة

```
public class Exemple
    {
        Type1 Attribut;//          المتغير الأول
        Type2 Attribut2;//      المتغير الثاني
        Type3 Attribut;//      المتغير الثالث

        Type Methode1()//  الدالة أو الإجراء الأول
           {
           }
        Type Methode2()//    الدالة أو الإجراء الثاني
           {
           }

    }
```

متغيرات الفئة هي كل الصفات التي تتميز بها ، والدوال و الإجراءات هي المهام التي تقوم بها كائنات الفئة، والفئة هي الصورة الأصلية لكل الكائنات التي لها نفس سلوكها و مظهرها، فإن أخذنا على سبيل المثال الكتب بأنواعها فهي تشترك في أنها مرتبطة بكاتب معين و عنوان و دار نشر و غير ذلك ، لذا يمكننا أن نعتبر الكتاب فئة ونعتبر "كليلة و دمنة" كائنا منسوخا من هذه الفئة.



كما يمكننا أن نأخذ الأستاذ كفئة شاملة ، و الأستاذ "محمد الباهي " كائنا لهذه الفئة لأن له متغيرات الأستاذ و سلوكاته.

ويمكننا مما سبق أن نقوم بإنشاء الفئة "كتاب" بمتغيراتها و مهامها ، ثم نقوم باستنساخ كائن منها:

```
class Livre
    {
     private string Titre;
     private string Auteur;
     private double Prix;

    public  void Initialiser(string titre, string auteur, double prix)
      {
          this.Titre = titre;
          this.Auteur = auteur;
          this.Prix = prix;
      }

     public void Information()
      {
          Console.Write("Le titre de livre est :{0}, son auteur est
:{1}",Titre,Auteur);
      }

    }
```

المتغيرات الثلاثة Titre و Auteur و Prix هي محددات الفئة Livre أي صفاتها، والدالة Initialiser التي تستقبل ثلاثة قيم دورها هو تحويل القيم المدخلة إلى متغيرات الفئة، أي إذا أعطينا للدالة Initialiser عند متغيرها الداخلي titre القيمة "Ryad Assalihin" فإن متغير الفئة Titre سيأخذ هذه القيمة و نفس الشيء بالنسبة لباقي المتغيرات، إذن فدور الدالة Initialiser هو إعطاء قيم لمتغيرات الكائن المنسوخ من الفئة. لا تخلط بين متغيرات الفئة ومتغيرات الدالة فالأولى تبدأ بحرف كبير و متغيرات الدالة مكتوبة كلها بحروف صغيرة.

أما الدالة Information فهي تقوم بإظهار قيم المتغيرات على الشاشة، لكن هذه الدالة لا تعمل إلا بعد المناداة على الدالة initialiser لأنها ستظهر قيما فارغة (طبيعي لأننا لم ننادي على الدالة Initialiser من أجل إعطاء قيم للمتغيرات).

ملاحظة:

الكلمة **This** تعني الفئة الحالية التي نشتغل عليها.

سنستعمل هذه الفئة في برنامجنا ، وللإعلان عن كائن منها سنقوم ب:

```
Livre MonLivre = new Livre();
```

حيث Livre هو اسم الفئة و الكلمة **new** تعني نسخة (instance) من الفئة.

ولولوج دوال الفئة نستعمل الكائن MonLivre لأنه نسخة من الفئة ، أي أن الشفرة ستكون هكذا :

```
using System ;
class Test
    {
        static void Main()
        {
            Livre MonLivre = new Livre();
            MonLivre.Initialiser("Kalila wa dimna", "Abdulah bno
lmoqafaa",75);
            MonLivre.Information();
            Console.ReadKey();
        }
    }
```

الدالة Initialiser تستقبل القيم الثلاثة مع وجوب احترام الترتيب وتخزنهم في متغيرات الكائن ثم تقوم الدالة Information بطباعة النتيجة على الشاشة.

عند تنفيذ هذه الشفرة ستكون النتيجة :

```
Le titre de livre est :kalila wa dimna,son auteur est : Abdulah bno
lmoqafaa
```

1. حدود تعريف الكائنات البرمجية (visibility):

لعلك لاحظت كثرة استعمالنا للكلمات public و static و غيرها ولعلك أخذت فكرة على معنى كل كلمة من هذه الكلمات، فهي التي تحدد لنا مدى قدرتنا إلى الوصول إلى الكائن البرمجي سواء أكان متغيرا أم دالة ... ، وهذه الكلمات سنلخصها في هذا الجدول لتعرف وقت استعمال كل واحدة منها:

التعريف الإفتراضي(طريقة الإعلان العادية)	الكائنات البرمجية معروفة فقط على مستوى الفئة التي تنتمي إليها.
Public	الكائنات البرمجية معروفة في كل فئات المشروع
Private	الكائنات البرمجية معروفة فقط على مستوى الفئة التي تنتمي إليها.
Protected	الكائنات البرمجية معروفة على مستوى الفئة فقط والفئات التي ترث منها.(سنرى مفهوم الوراثة قريبا إن شاء الله).
Internal	الكائنات البرمجية معروفة فقط على مستوى الفئات المنتمية إلى نفس assembly ، سنعرف كل هذا بالتفصيل إن شاء الله.

2. المجمعات assemblies

المجمع assembly عبارة عن ملف يضم كل ما يوجد في برنامجك من ملفات و فئات، ويكون امتداده إما exe أو dll حسب نوع المشروع، إن كان تطبيقا Application كان امتداد الأسمبلي exe وإن كان نوع المشروع مكتبة Bibliothéque كان امتداده dll.

3. مجالات الأسماء Les espaces des noms(namespaces)

وهي تلك الأسماء التي تكون مسبوقة بالموجهة using و هي مجالات تضم العديد من الفئات والأنواع، كما يمكن لمجال واحد أن يضم مجموعة من مجالات الأسماء كما هو الحال مع المجال System الذي نجد به فئات كثيرة مثل console و convert وكذلك يضم مجالات أسماء فرعية مثل IO و Collections وأول سطر يكون في البرنامج هو سطر التأشير إلى مجالات الأسماء ويكون باستعمال الأمر الموجه using متبوعا باسم مجال الأسماء ، واستعمال هذه الطريقة يوفر علينا أن نقوم كل مرة بكتابة مجال الأسم قبل فئاته، فلولا هذه الإمكانية لكتبنا System.Console.WriteLine() بدل Console.WriteLine().

عود على بدء:



كان لابد أن نقوم بالخروج عن موضوعنا الرئيسي لتوضيح هذه الأمور التي أرى أنه من الضروري أن تستوعبها حتى تتمكن من هضم مفهوم البرمجة الشيئية، كنا قد توقفنا عند إنشاء الفئات و استعمالها و الآن سوف نرى بحول الله امتدادات البرمجة الشيئية.

4. استنساخ الفئات instanciation

أي إنشاء كائنات جديدة منسوخة من الفئة الأصلية ويكون ذلك باستعمال الكلمة New :

```
Personne Person = new Personne();
```

5. استعمال static

المتغيرات و الدوال التي تكون مسبوقة بهذه الكلمة يمكننا استعمالها من غير استنساخ للفئة أي استعمالها مباشرة. كما يعرض هذا المثال :

```
using System;
class Personne
{
    static public  int Age;
    static public int returnAge()
    {
        return Age;
    }
}


class Test
{
    static void Main()
    {
        Personne.Age = 21;
        int Age = Personne.returnAge();
        Console.WriteLine(Age);
        Console.ReadKey();
    }
}
```

شاهد كيف قمنا باستعمال المتغير age و الدالة returnAge بلا إنشاء نسخة من الفئة Personne وذلك عن طريق Static.



6. المشيدات Constructors

المشيد هو عبارة عن دالة تحمل نفس اسم الفئة لكنها لا تعيد شيئا، وهي أول جزء ينفذ عند إنشاء كائن من فئة معينة، دوره يتجلى في إعطاء قيم بدئية لمتغيرات الفئة كما رأينا سابقا مع الدالة Inistialiser ، وهو لا يحتاج على أن نعلن عن نوعه كما هو الحال مع الدوال العادية بل لا يقبل حتى استعمال void.

إن كان المشيد ينتظر قيما ليعطيها لمتغيرات الفئة فإن الإعلان عن نسخة من هذه الفئة يكون كالتالي :

```
class object=new class(arg1,arg2,....,argN);
```

إن كانت الفئة تحتوي على مجموعة من المشيدات فإن الإعلان عن كائن من هذه الفئة يستلزم التعامل مع مشيد على الأقل و إن لم تكن الفئة تحتوي قط على أي مشيد فإن الإعلان يكون عاديا، ولكن ليكن في علمك أن السي شارب تستعمل مشيدا افتراضيا بدون متغيرات داخلية لا نراه و لكن نستعمله.

وخير دليل على ذلك هو طريقة الاستنساخ instanciation :

```
class object=new class();
```

ولتجميع ما رأينا إلى حد الآن سوف ننشئ فئة جديدة نسميها منتوج Article وسنقوم بتزويدها بمتغيرات داخلية attributes و دوال و طرق Methodes و كذلك بمشيدات constructors ، قبل ذلك سنناقش الأمر من أجل تدعيم مفهومنا للبرمجة الشيئية، من صفات المنتوج والتي سنعتبرها متغيرات داخلية: الرقم Code ، الثمن Prix، والنوع Type. و سننشئ دالة تمكننا من حساب ثمن كمية منتوجات معينة ، وكذلك مشيد من أجل إعطاء قيم لمتغيرات الفئة. (**حاول أن تعتبره تمرينا قبل أن تنظر إلى الحل لكي تقارن و تستفيد أكثر**).

```
using System.Text;
    class Article
    {
        //Attributes   المتغيرات الداخلية
        private int Code;
        private string Type;
        private double Prix;

        //Constructor  المشيد
        public Article(int code, string type, double prix)
        {
            this.Code=code;
            this.Type = type;
            this.Prix = prix;
         }

        //Methode     الدالة الحسابية
        public double CalculPrix(int Quantite)
        {
            double Montant;
            Montant = Quantite * Prix;
            return Montant;
        }

    }
```

المشيد يستقبل القيم الثلاثة و يعطيها لمتغيرات الفئة، أما الدالة CalculPrix فهي تستقبل الكمية المطلوبة من المنتوج وتحسب ثمنها Montant ، هذا الأخير تساوي قيمته جداء الكمية المطلوبة مع ثمن المنتوج.

لاستعمال هذه الفئة انظر هذا المثال التجريبي :

```
    using System;
    class Test
    {
        static void Main()
        {
            Article MonArticle = new Article(1, "Ordinateur", 4500);
            double Montant = MonArticle.CalculPrix(4);
            Console.Write("Le montant de cet article est  : " + Montant);
            Console.ReadKey();
        }

    }
```

قمنا في أول سطر باستنساخ كائن من الفئة سميناه MonArticle وأعطيناه قيمه عن طريق المشيد (لاحظ أن المشيد هو أول شيئ نستعمله عند الاستنساخ)، ثم قمنا بالإعلان عن متغير من نوع عشري وأعطيناه قيمة الدالة CalculMontant التي ولجنا إليها عن طريق الكائن MonArticle وأعطيناه القيمة 4 أي أننا نريد حساب ثمن أربعة منتوجات من صنف الكائن، ثم في الأخير أظهرنا النتيجة على الشاشة.

7. خصائص الفئات Properties

وهي طرق تستعمل من أجل التعامل مع المتغيرات الداخلية للفئة كما لو أنها عامة أي تمكننا من الولوج إليها لقراءة قيمها وللتعديل عليها، معتمدة على الأمرين **get** ويستعمل لإرجاع قيمة المتغير و كذلك الأمر **set** ويستعمل من أجل التعديل على قيمة المتغير و صيغتها العامة كما يلي :

```
public Type Property

        {

            get { return Attribute; }
            set { Attribute = value; }
        }
```

حيث Type هو نوع الخاصية و ينبغي أن يكون من نفس نوع المتغير ، Property هو اسم الخاصية و الأمر get من أجل تحديد المتغير المراد إرجاع قيمته، و الأمر set من أجل التعديل عليه وإعطائه قيمة جديدة و كذلك القيام بتحقق ما.

ولتبيان كيفية استعمال الخصائص سنوردها في المثال السابق، حيث سننشئ خاصية للمتغير الداخلي Code سنسميها PropertyCode ونفس الشيئ بالنسبة لباقي المتغيرات الداخلية ، وللإشارة فقط فإن إعطاء نفس الإسم للمتغير و خاصيته مرفوض لأنه سيحدث تعارض بينهما على مستوى الفئة.

ثم سنرفق بعد ذلك مثالا لكيفية استغلال الخصائص و المناداة عليها:




```
using System.Text;
 class Article
 {
     //Attributes         المتغيرات الداخلية
     private int Code;
     private string Type;
     private double Prix;

     //Constructor              المشيد
     public Article(int code, string type, double prix)
     {
         this.Code=code;
         this.Type = type;
         this.Prix = prix;
      }

     //Methode                   الدالة الحسابية
     public double CalculPrix(int Quantite)
     {
         double Montant;
         Montant = Quantite * Prix;
         return Montant;
     }

     //Properties               الخاصيات
     public int PropertyCode
     {
         get { return Code; }
         set { Code = value; }
     }

     public string PropertyType
     {
         get { return Type; }
         set { Type = value; }
     }

     public double PropertyPrix
     {
         get { return Prix; }
         set { Prix = value; }
     }
 }
```

```
using System;
class Test
{
    static void Main()
    {
        Article MonArticle = new Article(1, "Ordinateur", 4500);
        Console.Write("Le code de l\'article est :{0},"
        +"son type est :{1},son prix unitaire est:{2}"
        ,MonArticle.PropertyCode,MonArticle.PropertyType,
        MonArticle.PropertyPrix );
        Console.ReadKey();
    }

}
```

لقد قمنا بإظهار قيم المتغيرات عن طريق الخاصيات ، لحد الآن لم نستعمل سوى الجزء **get** من الخاصيات وفي هذا المثال سوف نقوم بالإستفادة من خدمات **set** التي قلنا بأنها تمكننا من التعديل على قيمة المتغير:

```
using System;
class Test
{
    static void Main()
    {
        Article MonArticle = new Article(1, "Ordinateur", 4500);
        MonArticle.PropertyCode = 100;
        MonArticle.PropertyType = "Télévision";
        MonArticle.PropertyPrix = 5000;
        Console.Write("Le code de l\'article est :{0},"
        + "son type est :{1},son prix unitaire est:{2}"
        , MonArticle.PropertyCode, MonArticle.PropertyType,
        MonArticle.PropertyPrix);
        Console.ReadKey();
    }

}
```

الآن أتوقع منك أن تعرف النتيجة بدون تنفيذ البرنامج ، حتما سيظهر لنا البرنامج القيم الثانية لأننا قمنا بالتعديل على قيم متغيرات الفئة باستعمال الخاصيات و هذا هو دور الأمر **set** .

بإمكاننا أن نستنتج الآن أن المتغير وخاصيته شيئ واحد وأن أي تغيير يمس أحدهما يطال الآخر.

ويمكننا أيضا التحقق من القيم المدخلة عن طريق استعمال الخاصيات كما ترينا هذه الخاصة :

```
    public double PropertyPrix
    {
        get { return Prix; }
        set {
            if (value == 0)Console.WriteLine("Le prix doit être"
                                          +" supérieur à 0");
            else
            Prix = value;
            Console.WriteLine("Prix accepté !!!");
            }
    }
```

هذه الخاصية تتحقق من القيمة التي نعطيها للمتغير Prix فإن كانت تساوي 0 أظهرنا رسالة تحذير وإن كان العكس قبلنا القيمة وأظهرنا رسالة إيجابية.

ويمكننا استعمال هذه الخاصية بنفس الطريقة السابقة وهي ستقوم بالتحقق من مجرد المناداة عليها ، كما يعلمنا هذا المثال :

```
using System;
class Test
{
    static void Main()
    {
        Article MonArticle = new Article(1,"Ordinateur",4500);
        Console.WriteLine("Donner le nouveau prix de l\'article");
        MonArticle.PropertyPrix = Convert.ToDouble(Console.ReadLine());
        Console.ReadKey();
    }
}
```

لأن نوع الخاصية PropertyPrix عشري قمنا بتحويل الدالة ReadLine() إلى نوع عشري.

وعلى ذكر تقنيات التحقق من القيم المدخلة فإنه يبدو أنه حان الوقت لنتعرف على كيفية التعامل مع الأخطاء.

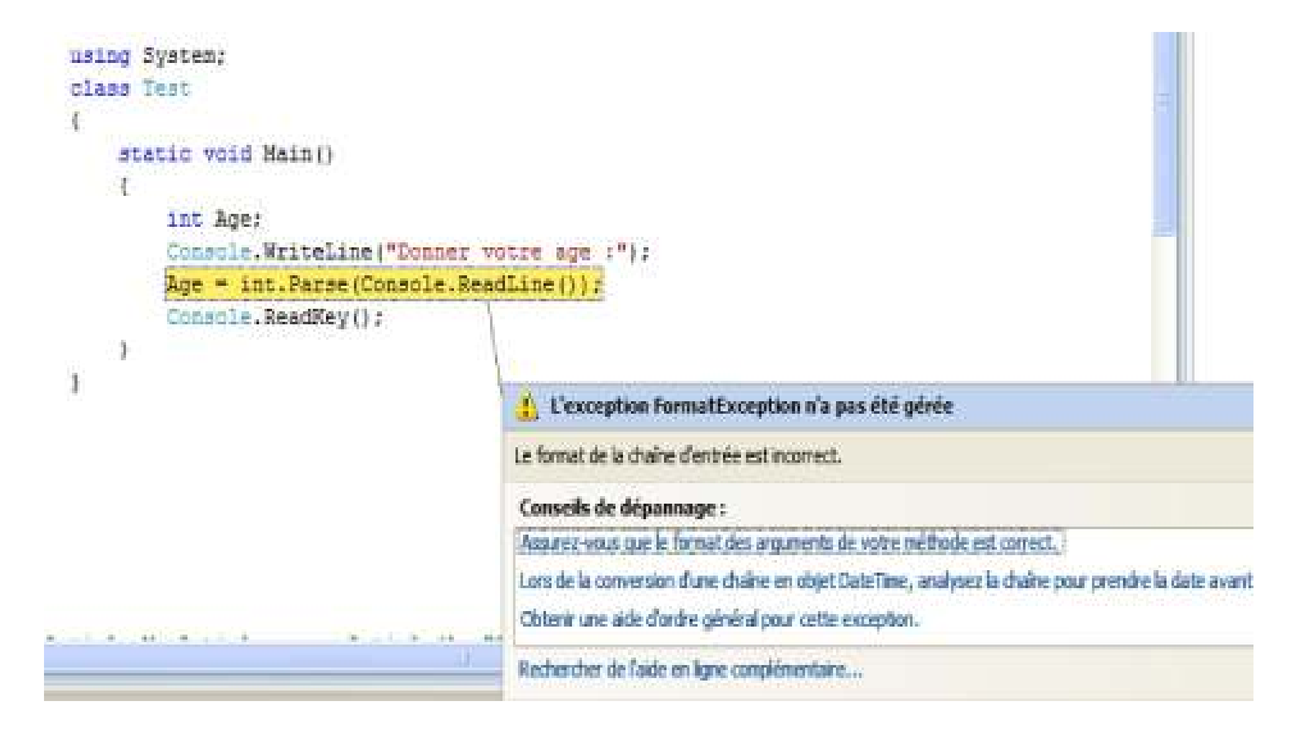
8. التعامل مع الأخطاء :

لنتأمل هذا المثال السهل لنبدأ من خلاله تعرفنا على معالجة الأخطاء:




```
using System;
class Test
{
    static void Main()
    {
        int Age;
        Console.WriteLine("Donner votre age :");
        Age = int.Parse(Console.ReadLine());
        Console.ReadKey();
    }
}
```

لنفترض أن المستعمل قام بإدخال قيمة غير رقمية، ماذا تتوقع أن يحدث لبرنامجك ؟ أجل سوف يختل وسيظهر لنا هذه الرسالة التحذيرية:



مضمون هذه الرسالة أن القيمة التي قمنا بإدخالها غير صحيحة، أي أن نوعها يخالف نوع المتغير الذي يتلقاها،وقد تحدث أخطاء أخرى غير هذه ، ولتفادي مثل هذه الأمور تمنحنا السي شارب هذه البنية لمعالجة القيم المدخلة وغيرها :

```
try
{
    //instructions
}
catch
{
    //resultats
}
```

في الجزء الأول نكتب العبارة التي نرغب في التحقق منها ، وفي الجزء الثاني النتائج التي سنعطيها للمستعمل في حالة وقوع الخطأ،كأن نظهر له رسالة تحذيرية،سنستغل هذه البنية في مثالنا السابق :

```
using System;
class Test
{
    static void Main()
    {
        int Age;
        Console.WriteLine("Donner votre age :");
        try
        {
            Age = int.Parse(Console.ReadLine());
        }
        catch
        {
            Console.WriteLine("Il faut saisir une valeur numérique
                               !!!!");
        }
        Console.ReadKey();
    }
}
```

في هذه الحالة إذا قام المستعمل بإدخال قيمة غير رقمية ، سيظهر البرنامج رسالة تحذيرية له ولن يحدث أي خطأ.

9. مصفوفة من الكائنات :

يمكننا أن ننشئ مصفوفة تضم العديد من النسخ بالقدر الذي نحتاجه بنفس الطريقة التي رأيناها سابقا، لكننا سنضيف قبل ذلك دالة اسمها Information تقوم بطباعة معلومات الكائن،سنستعمل نفس المثال السابق (الفئة منتوج Article):

```
    public void Information()
    {
        Console.WriteLine("Le Code de l\'article est :{0}"
            + ", Son Type est :{1},et son Prix est :{2}"
            + "", this.Code, this.Type, this.Prix);
    }
```

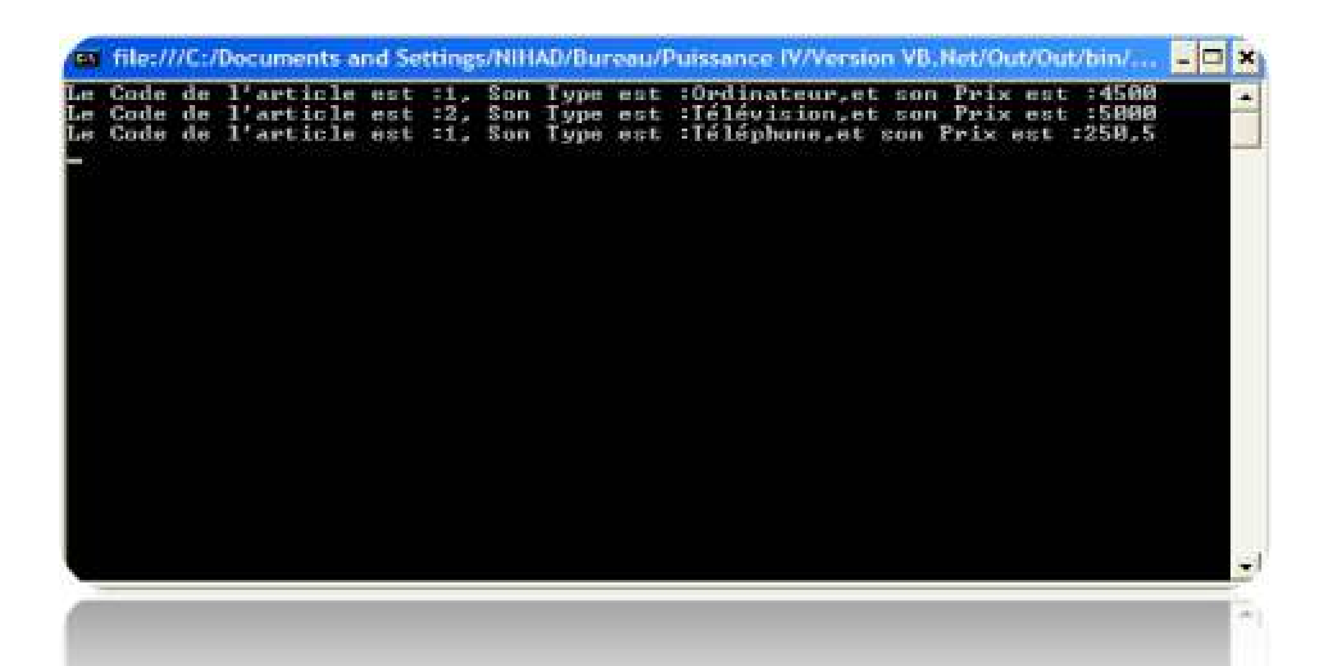
وهذا مثال سننشئ فيه مصفوفة من الكائنات وسنظهرها جميعا باستعمال هذه الدالة، وبالإعتماد على البنية التكرارية:

```
using System;
class Test
{
    static void Main()
    {

        Article[] MonArticle = new Article[3];
        MonArticle[0] = new Article(1, "Ordinateur", 4500);
        MonArticle[1] = new Article(2, "Télévision", 5000);
        MonArticle[2] = new Article(1, "Téléphone", 250.50);
         for (int i = 0; i<MonArticle.Length; i++)
         {
             MonArticle[i].Information();
         }
         Console.ReadKey();
    }

}
```

أنشأنا ثلاث نسخ في آن واحد بالإعتماد على المصفوفة،ثم أعطينا لكل كائن قيمه،ثم قمنا بتكرار ينطلق من بداية المصفوفة أي من أول كائن إلى طولها أي إلى آخر كائن،ثم قمنا بالمناداة على الدالة Information التي قمنا بإنشائها للتو من أجل طباعة قيم الكائنات على الشاشة ،عند التنفيذ ستكون النتيجة كالتالي :



10. تعدد التعاريف overloading (la surcharge)

نتحدث عن تعدد التعاريف عندما تتوفر إحدى الفئات على مجموعة من الدوال و الإجراءات التي لها نفس الإسم ، لكنها تختلف في المتغيرات الداخلية و أحيانا النوع الذي تعيده.

وهذا المفهوم يستعمل بكثرة في البرمجة المتطورة حيث يسمح للمطور أن يوفر عليه عناء تغيير الأسماء وهذه الصورة تعرض لنا إحدى الفئات التي تطبق هذا المفهوم :

```
class Program
{
    static void Main(string[] args)
    {
        string Name;
        Name=Convert.ToString(|
    }        ▲ 3 sur 36 ▼  string Convert.ToString (char value)
}            value: Caractère Unicode.
```

الدالة ()ToString الموجودة ضمن الفئة Convert معددة 36 مرة مع اختلاف المتغيرات الداخلية.

وحتى تتضح الصورة تأمل هذا المثال :

```
using System;
class Ouvrier
{
    //    متغيرات الفئة
    private int ID;
    private string Nom;

    //  تعدد التعاريف بالنسبة للمشيدات
    public Ouvrier()
    {
    }
    public Ouvrier(int n1, string n2)
    {
        this.N1 = ID;
        this.N2 = Nom;
    }

    //  تعدد التعاريف بالنسبة للدوال
    public void SearchInfo(int ID)
    {
        //
    }
    public void SearchInfo(string Nom)
    {
        //
    }
}
```

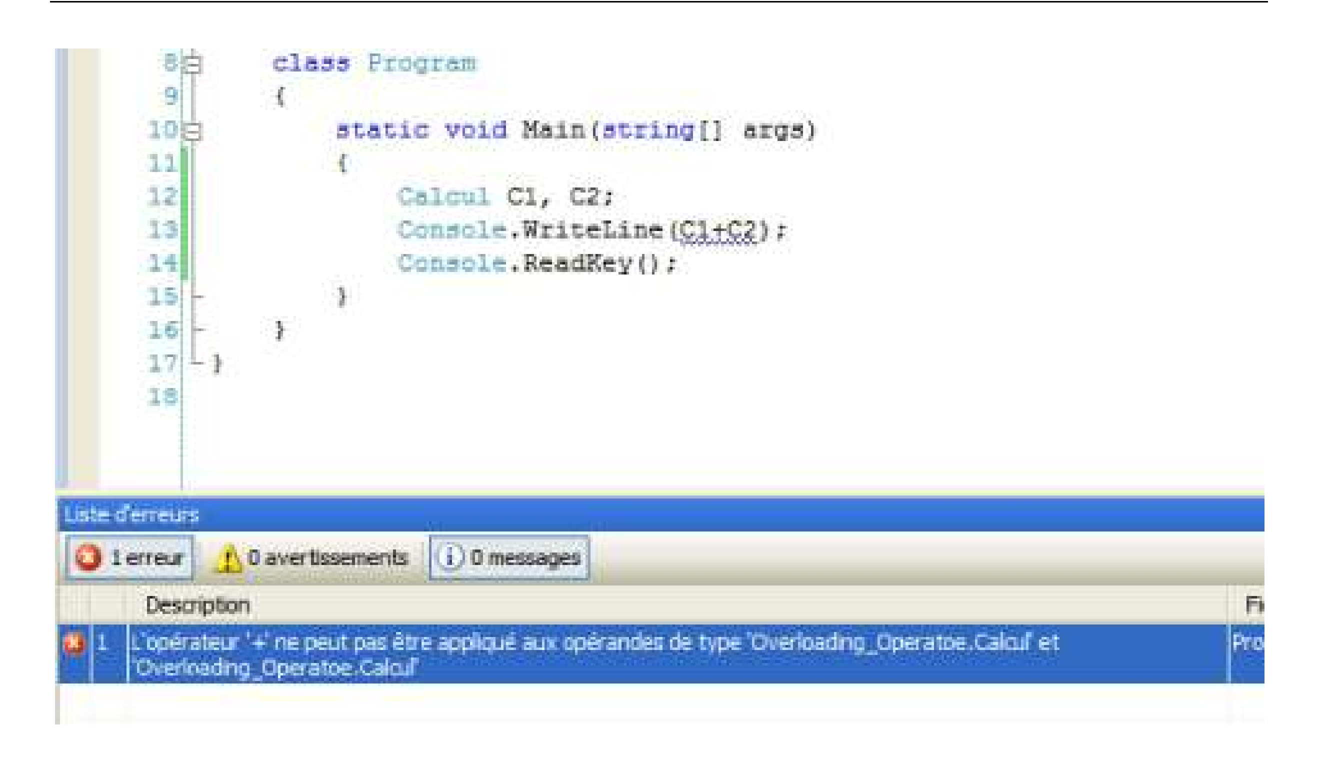
يمكن عمل تعدد التعاريف حتى بالنسبة للمشيدات، ينبغي فقط تغيير متغيرات المشيدات الداخلية ، وفيما يخص الدوال فقمنا هنا بعمل تعدد تعاريف للدالة `SearchInfo` حيث يمكننا أن نبحث عن معلومات العامل إما عن طريق رقمه الخاص وكذلك عن طريق اسمه ، لذا لا يبدو لائقا أن نستعمل دالتين مختلفتين اسميا.

أتمنى أن تدرك الغاية من استعمال هذه الطريقة و أن تحاول إقحامها في الوقت المناسب .

11. تعدد التعاريف بالنسبة للروابط Operators overloading:

إذا حاولت أن تقوم بعملية من العمليات الحسابية على الكائنات،فإن المترجم سيمنعك وسيعطيك رسالة الخطأ التالية بحجة أنه لا يمكن القيام بالعمليات على الكائنات:




```
 8  class Program
 9  {
10      static void Main(string[] args)
11      {
12          Calcul C1, C2;
13          Console.WriteLine(C1+C2);
14          Console.ReadKey();
15      }
16  }
17  }
18
```


لكن لاتيأس فبفعل تعدد التعاريف ستتمكن من إنشاء روابطنا الخاصة حيث سيصير بإمكاننا القيام بالعمليات على كائناتنا كأنها أرقام أو نصوص .

سنأخذ على سبيل المثال فئة الأعداد العقدية Complexe، هذه الأخيرة التي تتكون من جزأين أحدهما حقيقي والآخر وهمي وهذا نموذج على مجموعة من الأعداد العقدية :

```
3+2i,-5+12i,12+i ,…
```

الجزء الحقيقي هو الذي على اليمين ويكون إما موجبا أو سالبا، والجزء الوهمي أو الخيالي هو الذي عن اليسار والذي يكون مرتبطا مع العدد i ، لا يهم أن تعرف الأعداد العقدية لأننا لسنا في درس الرياضيات ولكن الأهم أن تفهم بنيتها حتى نصل إلى جوهر مفهوم إعادة تعريف الروابط.

والآن دونك هذه الفئة التي تقوم بإعادة تعريف الرابط الحسابي + لجمع عددين عقديين:




```
using System;
class Complexe
    {
        //( متغيرات الفئة (الجزء الحقيقي و الجزء الخيالي
        private int Reel, Imaginaire;

        //مشيد بمتغيرات داخلية من أجل إعطاء قيم لمتغيرات الفئة
        public Complexe(int reel, int imaginaire)
        {
            this.Reel = reel;
            this.Imaginaire = imaginaire;
        }
        // + إعادة تعريف الرابط
        public static Complexe operator +(Complexe C1, Complexe C2)
        {
          return new Complexe(C1.Reel + C1. Reel, C2. Imaginaire +
C2.Imaginaire);
        }
        // دالة من أجل إظهار النتيجة على شكل عدد عقدي
        public string Affichage()
        {
            return(String.Format("{0}+{1}i",Reel,Imaginaire));
        }
    }
```

للفئة Complexe متغيرين داخليين وهما الجزء الوهمي والحقيقي، أما دالتنا المهمة وهي التي تقوم بإعادة تعريف الرابط + فبنيتها تتكون من نوع الرجوع وهو الفئة Complexe ، وكذلك رمز الرابط مسبوقا بالكلمة **operator**، وبين القوسين أعلنا عن متغيرين داخليين من نوع الفئة وهما من سنقوم بإجراء العملية عليهما حيث ستعيد الدالة مجموع هذين الكائنين .

أما الدالة Affichage فإنها لا تقوم سوى بالعناية بطريقة العرض،حيث تعيد لنا قيمة نصية تحتوي على الجزء الصحيح مدموجا مع الجزء الخيالي .

حاول أن تركز على المثال بعناية فمن خلاله ستفهم جيدا خاصية إعادة التعريف، وهذا مثال على استعمال هذه الفئة :




```
using System;
 class Program
 {
     static void Main(string[] args)
     {
         Complexe c1 = new Complexe(3, 5);
         Complexe c2=new Complexe(4,6);
         Console.WriteLine("Le premier nombre complexe est:{0} , et le
         deuxiéme est :{1}",c1.Affichage(),c2.Affichage());
         Console.WriteLine("La somme de ces deux complexes est " + (c1 +
         c2).Affichage());
         Console.ReadKey();
     }
 }
```

بالإعتماد على هذه الطريقة يمكننا أن نقوم بإعادة تعريف باقي الروابط باستثناء :

- الروابط == و =! فينبغي إعادة تعريفهما في نفس الوقت.
- الروابط && و || و =- و =+ ... لا يمكن إعادة تعريفهم.

12. الوراثة L'héritage (Inheritance)

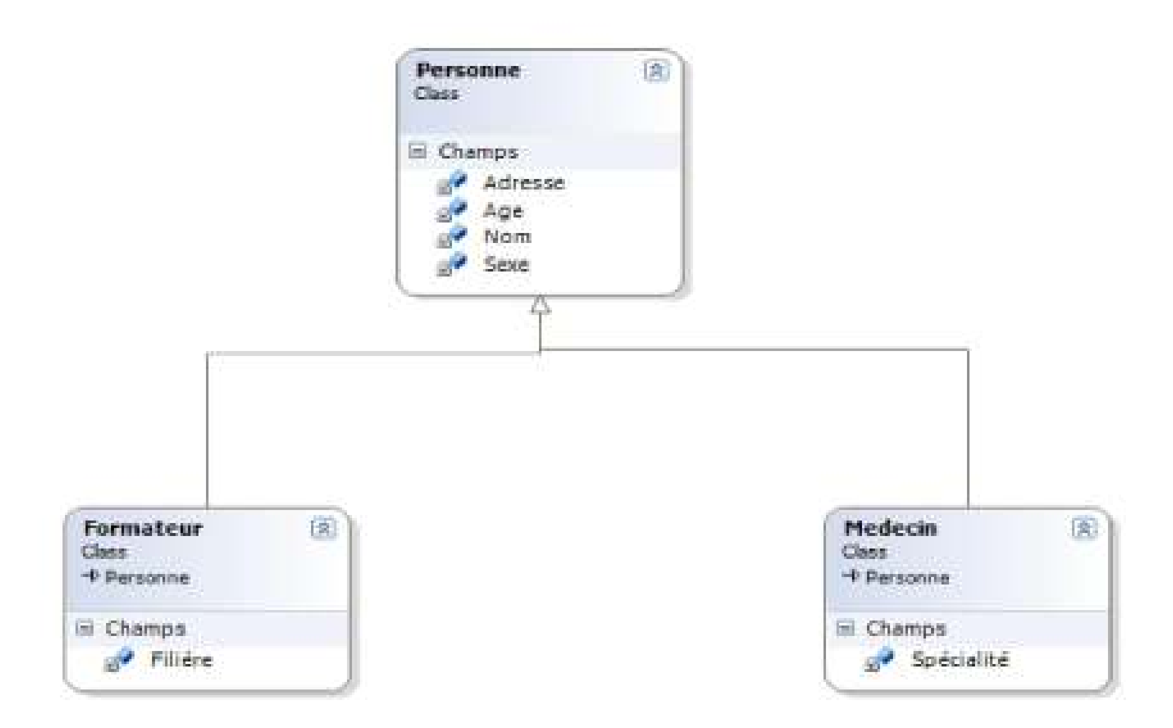
من المفاهيم الأساسية ومضمونها أن فئة تمتلك مجموعة من الخصائص يمكن توريثها إلى فئات أخرى، بمعنى آخر توجد الفئة الأم وهي تلك التي تملك الصفات الرئيسية، والفئة البنت التي ترث هذه الصفات .

والغاية من الوراثة هي تخليص المبرمج من إعادة كتابة الأكواد المشتركة بين الفئات المتوارثة ، حيث يكفي أن يقوم بإنشاء الفئة الأم ثم يرث دوالها ومتغيراتها في الفئات المشتقة ، ثم يقوم بإضافة الصفات الخاصة لهذه الفئات.

توجد أمثلة كثيرة عن الفئات المتوارثة، و كنموذج للتوضيح سنأخذ الفئة (شخص Personne) التي تملك مجموعة من الصفات الأساسية كالإسم ، العمر ، الجنس .. ، وسنعتبرها الفئة الأم ، وسنورث صفاتها إلى فئات أخرى مثلا الفئة (طبيب Medecin) و الفئة (معلم Formateur) فهما فئتان تتوفران على نفس خصائص الفئة Personne بالإضافة إلى صفاتهما الخاصة،الشيء الذي يمكن تمثيله على الشكل التالي :







وهذا هو الكود الخاص بالفئة Personne :

```
public class Personne
{
    private string Nom;
    private string Adresse;
    private string Sexe;
    private short Age;
}
```

للوراثة من هذه الفئة سنحتاج فقط إلى إضافة نقطتين (:) أمام إسم الفئة البنت متبوعة بإسم الفئة الأم، أي ما يمكن ترجمته إلى الصيغة التالية:

```
public class ClassFille:ClasseMere
{

}
```

من خلال ما رأينا يمكننا أن نستنتج أن كود الفئتين Personne و Medecin هو :

```
// الفئة Medecin

public class Medecin : Personne
{
    private string Spécialité;
}
```

```
// الفئة Formateur

public class Formateur : Personne
{
    private string Filiére;
}
```

13. الفئات المجردة Les classes abstraites(Abstract classes)

الفئات المجردة هي عبارة عن فئات لا يمكن أن ننسخ منها كائنات،أي لا نستطيع البتة أن ننشئ كائنا من نوعها، بل ينبغي أن نقوم بعمل توريث لها وعمل استنساخ من الفئات المشتقة منها،باختصار للإعلان عن كائن من نوع فئة مجردة نحتاج إلى فئة أخرى مشتقة منها ، وهي ما يسمى في الفيجوال بسيك MustInherit ، ولإنشاء فئة مجردة يلزمنا إضافة الكلمة المفتاح Abstract أمام اسم الفئة .

وهذا المثال يدعم ما قمنا بشرحه للحال:

```
abstract class Personne
 {
     private string Nom;
     private string Adresse;
     private string Sexe;
     private short Age;
 }
```

إذا قمنا بالإعلان عن كائن من هذه الفئة سيظهر لنا خطأ يعلمنا أن هذه الفئة مجردة ولا يمكن استنساخها، لذا ينبغي أن ننشئ فئة ترث من الفئة Personne خصائصها، ثم نقوم بالإعلان عن متغيرات من صنف الفئة البنت، وضمنيا ستكون للكائن خصائص الفئة الأم.

```
class Formateur:Personne
{
    private string Filiére;
}
```

الآن يمكننا أن ننشئ كائنات من الفئة Personne بكل حرية.

14. الفئات المغلقة Les classes scellés (Sealed classes)

تكون معرفة بالكلمة المفتاح sealed وهي فئات لا يمكن الوراثة منها ،سيعطيك المترجم خطأ إذا قمت بعمل اشتقاق فئة من إحدى الفئات من هذا النوع.الفئات المغلقة لا يمكن التعامل معها كالفئات المجردة.

```
sealed class Personne
 {

 }

 // sealed سينتج عن هذه الوراثة خطأ لأن الفئة الأم معلن عنها ب

class Medecin : Personne
{

}
```

15. الدوال الوهمية Les methodes virtuelles (Virtual Methods)

عند تطبيق مفهوم الوراثة بين الفئات، نحتاج أحيانا إلى القيام بإعادة تعريف بعض الدوال على مستوى الفئات المشتقة، نظرا لأن الدالة الأصلية لا تتلاءم مع متطلبات الفئة البنت أو تحتاج إلى تعديل لتتوافق مع المرغوب فيه، فمثلا إذا قمنا بتعريف دالة على مستوى الفئة Personne(شخص) وسميناها Travail() (عمل) وقمنا باشتقاق فئة سميناها Ouvrier (عامل) من الفئة Personne فإن هذه الدالة تبقى مبهمة،لأن العامل نعرف مهنته.

لذلك سنحتاج إلى إعادة تعريف الدالة Travail() لتتوافق مع الفئة البنت Ouvrier .



هنا سنحتاج إلى استعمال الكلمة المفتاح Virtual قبل اسم الدالة Travail في الفئة الأم لنعلم المترجم بأن هذه الدالة سيعاد تعريفها و كذلك إلى الكلمة override التي سنكتبها قبل اسم الدالة على مستوى الفئات المشتقة. لنعلم المترجم بأن هذه الدالة سنعيد تعريفها.

```
class Personne
{
    string Nom;
    string Adresse;
    string Sexe;
    short Age;

    public virtual void Travail()
    {
        Console.WriteLine("Je travaille");
    }
}

class Ouvrier : Personne
{
    public override void Travail()
    {
        base.Travail();
        Console.WriteLine("Et plus précisément je suis un ouvrier");
    }
}
```

الكلمة base تعني أن الدالة التي تأتي من بعدها تعود إلى الفئة الأم، لاحظ المثال جيدا وحاول أن تضع فيه لمستك كي تستوعبه أكثر ، لن أضيف سوى أن النتيجة ستكون كما يلي :

```
Je travaille
Et plus précisément je suis un ouvrier
```



16. الكلمة new

لها تقريبا نفس دور الكلمة override ، حيث تقوم بإعادة تعريف دوال الفئة الأم ، إلا أنها تختلف عن override في كونها تقوم بإنشاء دالة جديدة تحمل نفس اسم الدالة الأم، حيث تقوم بإخفائها ولكن على مستوى الفئة البنت فقط.بحيث تبقى الدالة الأم هي نفسها لايطالها أي تعديل ، حتى ولو أعطينا كائنا من الفئة الأم قيمة كائن من الفئة البنت.

يبدو أنني أرهقتك بالحديث حول هذه العائلة الغامضة، لكن لا تخف ستفهم كلامي إذا تأملت هذا النموذج:

```
class Art
{
    Public virtual void Description()
    {
        Console.WriteLine("l\'art est la classe mére");
    }
}

class Theatre:Art
{
    new public void Description()
    {
        Console.WriteLine("Le théatre est une classe fille");
    }
}
```

والآن ليتضح الأمر انظر إلى هذا المثال :

```
using System;
class Test
{
    static void Main()
    {
        Theatre theatre = new Theatre();
        Art art = theatre;
        art.Description();
        Console.ReadKey();
    }
}
```



مع العلم أننا أعطينا الكائن المنسوخ من الفئة الأم Art الكائن الأبن كقيمة فهذا لا يؤثر بتاتا على الدالة الأصلية، لأنه يعتبر الدالة البنت المعلن عنها ب new دالة جديدة تختلف عن الدالة الأصلية، أي أن النتيجة ستكون كالآتي :

```
l'art est la classe mére
```

أردت إنهاء هذا الفصل لكن نفسي غير مرتاحة ، لأني أعتقد أن الأمر يحتاج إلى مثال متكامل يشرح الفرق بين new و override . فليكن الأمر كذلك لندرس هذا المثال :

```
//   هذه هي الفئة الأم

class Art
 {
     public virtual void Description()
     {
         Console.WriteLine("l\'art est la classe mére");
     }
 }

   // وهذه فئة مشتقة

 class Theatre:Art
 {
     new public void Description()
     {
         Console.WriteLine("Le théatre est une classe fille");
     }
 }

   // وهذه فئة مشتقة أخرى

class Cinema:Art
 {
     public override void Description()
     {
         Console.WriteLine("la cinéma est une classe fille");
     }
 }
```



لم نقم سوى بإضافة فئة مشتقة جديدة سميناها Cinema ، هذه الأخيرة تضم دالة تعيد تعريف الدالة الأصلية باستعمال override .

الآن أصبح لدينا فئتان مشتقتان: الأولى تستعمل الكلمة new و الثانية تستعمل الكلمة override ، الآن بقي لنا استغلال هذه الفئات لنتمكن من فهم الفرق بين هذين الكلمتين.

```
using System;
class Test
{
     static void Main()
     {
          Theatre theatre = new Theatre();
          Cinema cinema = new Cinema();


          Art art = new Theatre();
          Console.WriteLine("Ici on a utilisé <--new-->\n");
          art.Description();


          art = new Cinema();
          Console.WriteLine("Ici on a utilisé <--override-->\n");
          art.Description();
          Console.ReadKey();
     }
}
```

الحالة الأولى سيظهر نتيجة الدالة الأصلية لأننا استعملنا الكلمة new أما في الحالة الثانية فسيظهر نتيجة دالة الفئة Cinema لأننا أعلنا عنها باستعمال override .

```
Ici on a utilisé <--new-->

L'art est la classe mére'art est la classe mére

Ici on a utilisé <--override-->

la cinéma est une classe fille
```



17. تعدد الأشكال Le polymorphisme (Polymorphism)

لا ترتبك من هذا المصطلح، لأنك تعرفه جيدا لأننا تعاملنا معه سابقا كشريك مجهول والآن حان الوقت لنفصح عن هويته أو بالأحرى لنفضحه ! ! ! !

عندما تحدثنا عن الوراثة كنا قد قلنا بأن الفئات المشتقة ترث من الفئة الرئيسية كل عناصرها ، أي بمعنى آخر يمكننا التعامل مع الكائنات البنت كأنها كائنات منسوخة من الأب.

أرى أن الأمر قد اتضح الآن بقي فقط أن نسرد مثالا يثبت أننا فهمنا :

لتكن Sport فئة أم و Tennis فئة مشتقة منها و Amation() دالة معزولة عن الفئتين تتلقى متغيرا داخليا من نوع Sport :

```
static void Amation(Sport S)
{
    //
}
```

عند المناداة على هذه الدالة، فنحن لسنا مجبرين على أن نعطيها كائنا من نوع Sport بل بإمكاننا أن نعطيها أي كائن منسوخ من الفئات المشتقة :

```
using System;
class Program
{
    static void Main()
    {
        Teenis T = new Teenis();
        Amation(T);
    }
}
```

لا تخف فهذا الكود لن يحدث خطأ لأن الفئة Tennis ترث من الفئة Sport أو بطريقة أخرى صديقنا القديم (مجهول الهوية) قد أقحم نفسه هنا ليفرض وجوده.

18. الواجهات Les interfaces (Interfaces)

سأصدمك إن قلت لك بأنه لا يمكننا على الإطلاق عمل وراثة من مجموعة من الفئات الرئيسية، بحيث تتيح لنا السي شارب الوراثة من فئة واحدة فقط فهي لا تدعم الوراثة المتعددة، ولكنني سأعيد إليك الأمل عندما أقول لك بأن هناك بديلا لهذا النقص (ليس نقصا مئة في المئة لأنه يصعب أن نجد فئة ترث من أكثر من فئة في حياتنا).

هذا البديل يا أخي اسمه الواجهات interfaces وهي مثل الفئات ولكنها مجردة abstract بمعنى لا يمكن أن ننشئ منها نسخا ولا يمكننا كتابة أي كود فيها ما عدا الإعلان عن الدوال والطرق والخصائص التي ستحتاجها الفئات المستعملة لهذه الواجهة.

يمكن لفئة أن تستعمل (اعذروني على هذا المصطلح لأني لم أجد نظيرا ل implementation)، مجموعة من الواجهات في نفس الوقت.

وهذا نموذج لواجهة :

```
interface Quadrilatére
{
    int Longueur { get; set; }
    int Largeur{get;set;}
    float Surface();
}
```

سمينا الواجهة Quadrilatére أي مضلع رباعي (شكل له أربعة أضلاع) ، و أضفنا إليها الإعلان عن خاصيتين Properties (رأيناها سابقا) و دالة من نوع عشري سنحتاجها لحساب مساحة المضلع الرباعي طريقة استعمال الواجهات هي نفسها طريقة استعمال الوراثة بحيث سنضيف اسم الواجهة بعد نقطتين أمام اسم الفئة.

```
class Rectangle:Quadrilatére
{

}
```

والآن سوف نرى كيف نقحم عناصر الواجهة في الفئة التي تستعملها :

```
interface Quadrilatére
{
    int Longueur { get; set; }
    int Largeur{get;set;}
    float Surface();
}

class Rectangle:Quadrilatére
{
    private int Longueur;
    private int Largeur;

    public Rectangle(int longueur, int largeur)
    {
        this.Longueur = longueur;
        this.Largeur = largeur;
    }
    int Quadrilatére.Longueur
    {
        get { return Longueur; }
        set { Longueur = value;}
    }
    int Quadrilatére.Largeur
    {
        get { return Largeur; }
        set { Largeur = value;}
    }

    float Quadrilatére.Surface()
    {
        return Longueur * Largeur;
    }

}
```

حاول أن تتأمل المثال جيدا وشاهد طريقة استعمال عناصر الواجهة في الفئة Rectangle ، وابحث عن أمثلة أخرى من نسج تفكيرك لتعزز معارفك حول الواجهات لأننا سوف نغادرها توا ! !

وقبل ذلك سننشئ مثالا نستعمل فيه هذه الفئة لنستفيد أكثر فنحن هنا لنتعلم :






```
using System;
   class Test
   {
       static void Main()
       {
           Quadrilatére Q = new Rectangle(7, 6);
           Console.WriteLine(Q.Surface());
           Console.ReadKey();
       }
   }
```

الكائن Q من نوع الواجهة ولكنه منسوخ من الفئة Rectangle لأنه لا يمكن عمل استنساخ للواجهة على الإطلاق ، جرب و سترى ،ثم قمنا بطباعة مساحة المستطيل بالإعتماد على الدالة Surface().



19. المفوضات Les delegates (delegates)

إن استحضرت معي ما رأينا سابقا حول الأنواع بالمرجع و الأنواع بالقيمة ، سترى بأننا أوردنا هذا المصطلح خلف الظل و مررنا عليه مرور الكرام بل لم نوفه من حقه شيئا ما عدا ذكرنا له بأنه نوع بالمرجع.

أجل، فالمفوضات نوع بالمرجع و دورها هو التأشير إلى دالة أو إجراء معين ، إذ أن دورها هو تمرير الدوال و الإجراءات كأنها متغيرات داخلية، والغاية من المفوضات هو الإختيار السليم للدالة أو الإجراء المناسب في لحظة التنفيذ execution وليس في وقت الترجمة compilation. (ماذا تقول ؟ أتتكلم باليابانية؟)

ركز معي جيدا ، لنفترض أننا صنعنا برنامجا ولكن من نوع حدثي Evenementiel أي يتميز بالنوافذ و الأزرار و القوائم وغيرها ، وقام المستعمل بتشغيل البرنامج بمعنى أنه الآن في حالة تنفيذ، في هذه الحالة علينا أن نترك نائبا لنا في البرنامج أو بالأحرى مفوضا يتعرف على الأحداث التي يقوم بها المستعمل أي هل قام ب click أو double click أو قام بالضغط على زر ، لكي يتم ربط هذه الأحداث بالدوال و الإجراءات المناسبة لها ، هنا ستقول لي كيف سنستطيع ذلك ؟ سأقول لك بأن المفوضات في الخدمة !

للمفوضات دور جم حيث تسمح لك بتمرير الدوال و الإجراءات كأنها متغيرات داخلية شريطة أن تكون من نفس النوع الذي يعيده المفوض و أن تكون متغيراتها الداخلية مثل متغيرات المفوض.

وهذا مثال على كيفية الإعلان على مفوض delegate:

```
public delegate string myDelegate(string Texte);
```

حيث myDelegate هو اسم المفوض و string هو نوع الرجوع و Texte متغير داخلي، الآن كل دالة لها نفس بنية هذا المفوض يمكننا إسنادها إليه و سيتصرف تماما مثلها.

بعد الإعلان عن المفوض نحتاج إلى استنساخ كائن منه إذ سيصبح مفوضنا بمثابة فئة مشتقة من الفئة الأم system.delegate و كما تعلمون للتعامل مع الفئات يلزمنا استنساخ كائنات منها .

سيكون الإعلان كما يلي :

```
myDelegate instance = new myDelegate(Methode_à_passer);
```

كما يمكنك بكل بساطة أن تعلن عن الكائن هكذا (فقط إذا كنت تستعمل 2.0 #c فما فوق) :

```
myDelegate instance = Methode_à_passer;
```

الآن أصبح بإمكاننا أن نتعامل مع المفوض myDelegate تماما كأنه الدالة Methode_à_passer .

قبل أن نورد مثالا شاملا قل لي بصدق هل فهمت كيفية الإعلان عن المفوضات و استنساخها ، إن كان الأمر كذلك فاتبعني لنشق طريقنا بهذا المثال و إن كان العكس فإنك تعرف جيدا ما سأقوله لك (راجع الفقرة من بدايتها و اسأل الله أن يعينك).

هذا المثال سنحاول فيه جمع ما تعلمناه حول المفوضات :

أولا سنقوم بإنشاء فئة سنسميها testString و سننشئ فيها دالتين ، الأولى عادية والثانية static :

```
        class testString
        {

            public static string isSmall(string Texte)
            {
                if (Texte == Texte.ToUpper())

                    return "Le texte est majiscule";

                if (Texte == Texte.ToLower())

                    return "Le texte est miniscule";

                return "Le texte est mélangé";
            }


            public string Longueur(string Texte)
            {
                return string.Format("Le longueur de votre texte est :
{0}", Texte.Length);
            }

        }
```



الدالة الأولى تقوم بالتحقق من طبيعة حروف النص الذي سندخله أهي كبيرة أم صغيرة، و الدالة الثانية تعيد لنا طول النص أي عدد الحروف المكونة له.

الآن سنقوم بإنشاء المفوض :

```
public delegate string myDelegate(string Texte);
```

قمنا الآن بكل شيء، و لم يتبق سوى استنساخ المفوض و تمرير الدوال :

```
using System;

static void Main()
{
    testString testing = new testString();

    string Texte;

    myDelegate firstDelegate = new myDelegate(testString.isSmall);

    myDelegate secondDelegate = new myDelegate(testing.Longueur);

    Console.Write("Entrer votre texte :");

    Texte = Console.ReadLine();

    Console.WriteLine(firstDelegate(Texte));

    Console.WriteLine(secondDelegate(Texte));

    Console.ReadKey();
}
```

أنشأنا كائنين من المفوض، و مررنا للأول الدالة الثابتة isSmall حيث استعملناها مباشرة عن طريق الفئة، أما الكائن الثاني فأسندنا له الدالة العادية، ثم بعد ذلك طلبنا إدخال النص المراد إجراء الدالتين عليه ثم عالجناه بواسطة الكائنات المنسوخة من المفوض تماما كالدوال.

وهذا هو المثال بأكمله :




```
    class Delegates
    {

        public delegate string myDelegate(string Texte);


        class testString
        {

            public static string isSmall(string Texte)
            {
                if (Texte == Texte.ToUpper())
                    return "Le texte est majiscule";
                if (Texte == Texte.ToLower())
                    return "Le texte est miniscule";
                return "Le texte est mélangé";
            }


            public string Longueur(string Texte)
            {
                return string.Format("Le longueur de votre texte est :
{0}", Texte.Length);
            }
        }


        static void Main()
        {
            testString testing = new testString();
            string Texte;
            myDelegate firstDelegate = new myDelegate(testString.isSmall);
            myDelegate secondDelegate = new myDelegate(testing.Longueur);
            Console.Write("Entrer votre texte :");
            Texte = Console.ReadLine();
            Console.WriteLine(firstDelegate(Texte));
            Console.WriteLine(secondDelegate(Texte));
            Console.ReadKey();
        }
    }
```

20. التفويض المتعدد délégués multicast (multicast)

لحد الساعة المفوضات التي ذكرنا لا تقوم سوى بالتأشير إلى دالة واحدة أو إجراء واحد ، فما العمل إذن لو احتجنا إلى التأشير إلى أكثر من دالة أو إجراء ؟

ستقول لي بأننا لن نحتاج إلى هذا الأمر، لكنني سأعترض و أقول بأننا في البرمجة الحدثية نحتاج إلى هذا الأمر بشدة بحيث نرغب في تنفيذ مجموعة من الدوال و الإجراءات في نداء واحد ولن يتأتى لنا ذلك إلا باستعمال المفوضات المتعددة (هذا ما ستتعرف عليه في الفصل القادم إن شاء الله).

فعند الضغط مثلا على زر (في البرمجة الحدثية يسمى هذا الحدث كليك click) قد تحتاج إلى القيام بمجموعة من المهام.

نستطيع دمج العديد من المفوضات باستعمال الإشارة + وبالتالي سيتم النداء على كل الدوال و الإجراءات المررة إلى المفوضات المجموعة.

إن كنت من عشاق الرياضيات (وأنا لست منهم) فستكون على علم بدوال sinus و cosinus ،هذه الدوال سهلة الحساب باستعمال الفئة Math التي تضم العديد من الدوال الرياضية، لذلك سنستعملها كمثال على التفويض المتعدد ، حيث سننادي في الأول على كل دالة على حدى باستعمال المفوضات التي تعرفنا عليها سابقا ثم سنقوم بالنداء على الدالتين في آن واحد باستعمال التفويض المتعدد.

هذه هي الفئة التي تضم الإجراءين الخاصين بحساب sinus و cosinus :

```
class MultiCast
{

 public  static void Sinus(int Number)
    {
        Console.WriteLine("Le sinus de ce nombre est :" +
                          Math.Sin(Number));
    }

 public static void Cosinus(int Number)
    {
        Console.WriteLine("Le cosinus ce nombre est :" +
                          Math.Cos(Number));
    }

}
```

صراحة الفئة Math أراحتني من عناء العلاقات الرياضية لأنني بصراحة لا أحب ذلك النمط من التفكير ، مجرد رأي فردي فلا تعتدوا به ! ! !

والآن يا إخوتي سننشئ مفوضنا و سنشتق منه كائنين كما العادة و سنمرر لهما الدالتين السابقتين :

```
public delegate void MyDel(int Number);
static void Main()
{
    MyDel D1 = MultiCast.Sinus;
    MyDel D2=MultiCast.Cosinus;
    MyDel MultiCastDelegate = D1 + D2;
    Console.WriteLine("L\'appel de chaque methode:\n");
    D1.Invoke(60);
    D2.Invoke(60);
    Console.WriteLine("Et maintenant les deux appels en seule
                        fois:\n");
    MultiCastDelegate(60);
    Console.ReadKey();
}
```

لا تخف من منظر الدالة Invoke فيمكننا الإستغناء عنها كما عملنا في الفصل السابق ، أدرجتها فقط لتعرف دورها إذا التقيت بها في إحدى البرامج.

يمكننا أيضا القيام بعمليات الطرح على المفوضات و تكون بالرمز – لم أحب أن أسرد نموذجا لها تفاديا للتكرار فهي تعمل بنفس الطريقة التي رأينا للتو.

21. الأحداث Les événements (events)

عندما نضغط على زر أو نختار من قائمة أو نحرك الفأرة فإن ذلك يولد أحداثا تخول للبرنامج الذي نتعامل معه التعرف على مانروم القيام به.فعندما تضغط على زر حفظ في برنامج الوورد فإن المفوض المرتبط بهذا الحدث يستنتج بأننا نريد حفظ الوثيقة ولا شيء آخر فيقوم بالنداء على دالة الحفظ.

يتم الإعلان على الأحداث تماما كالمتغيرات مع إضافة الكلمة المفتاح event و أن يكون نوع الحدث مفوضا delegate.

نقوم في الأول بالإعلان عن المفوض، ثم نعلن عن الحدث المتعلق به ، وعندما نقوم بتهييج ذلك الحدث فإن المفوض يقوم بالمناداة على الدالة أو الإجراء الممرر. تسمى الدوال والإجراءات المرتبطة بحدث event handler.

سنقوم الآن بإنشاء فئة سنسميها Article وسنقوم بإعطائها متغيرا داخليا نسميه Quantite أي الكمية ثم نضيف مشيدا للفئة و خاصية Property للمتغير.

```
class Article
{
    //Attribut

    private int Quantite;

    //Constructeur

    public Article(int quantite)
    {
        this.Quantite = quantite;
    }


    //Propriété

    public int QuantiteProperty
    {
        get {
            return Quantite;
            }
        set {
            Quantite = value;
            }
    }
}
```

سنضيف إلى فئتنا الآن دالة تستقبل الكمية المطلوبة و تنقصها من الكمية المتاحة :

```
public int Commande(int NbrArticle)
{
    this.Quantite = Quantite - NbrArticle;
}
```

إذا طلب أحد الزبائن كمية أكبر من التي لدينا في المخزن ينبغي لبرنامجنا أن يظهر رسالة تعلم المستعمل بأن الكمية المطلوبة غير متاحة، فهنا يمكننا تطبيق مفهوم الأحداث فمثلا عند وقوع هذا الحدث علينا تنفيذ هذا الإجراء.

إذا استعملنا الفئة الآن كما هي فبإمكاننا طلب كمية أكبر من التي في المخزن ولن يمنعنا البرنامج لأننا لم نضف إليه الحدث المناسب.

فمثلا :

```
using System;
static void Main()
{
    Article article = new Article(20);
    article.Commande(60);
    Console.Write(article.QuantiteProperty);
    Console.ReadKey();
}
```

الكمية الموجودة عندنا من المنتوج هي 20 و الكمية المطلوبة هي 60 ، ينبغي أن يقوم البرنامج بمنعنا ولكن ذلك لن يقع لأنه سيقوم بعمليته الحسابية ليظهر لنا قيمة سلبية 40-.

الآن نحن في حاجة ماسة إلى إجراء يقع عند وقوع هذا الحدث، سنقوم أولا بإنشاء المفوض الذي سيربط الحدث مع الإجراء، ولنسمه commandeDelegate:

```
public delegate void commandDelegate();
```

وسنعلن عن الحدث من نوع هذا المفوض ولكن داخل الفئة ليكون معروفا:

```
public event commandDelegate commandEvent;
```

الآن قمنا بإنشاء الحدث بقي لنا التحقق من الكمية المطلوبة ثم إنشاء الإجراء الذي سيتم تنفيذه عند الحدث، سنذهب إلى دالتنا commande و نضيف هذا التحقق أولا :

```
public void Commande(int NbrArticle)
{
    if (NbrArticle > this.Quantite) commandEvent();
    this.Quantite  = Quantite  - NbrArticle;
}
```

مضمون التحقق أنه إذا كانت الكمية المطلوبة أكبر من الكمية الموجودة سنطلق الحدث الذي أنشأنا للتو، ولكنه فارغ ، سنقوم الآن بإنشاء الإجراء المناسب له:

```
public void messagaEvent()
{
    Console.Write("Rupture de stock !!!!");
}
```

الآن قمنا بكل شيء ، سنقوم فقط بإضافة هذا السطر ولكن خارج الفئة، الذي من خلاله سنربط هذا الإجراء مع حدثنا :

```
article.commandEvent += new commandDelegate(article.messagaEvent);
```

حيث article هو اسم الكائن المنسوخ من الفئة و commandDelegate هو اسم المفوض .
إذا قمنا بتنفيذ البرنامج و قيمة الكمية المطلوبة أكبر من الكمية الموجودة فإن الحدث سينادي على الإجراء messageEvent ليظهر لنا رسالة بأن المخزون غير كاف.

وهذا هو الكود العام للفئة Article و لفئة التجريب :

```
class Event
{
    class Article
    {
        //Evenement **** Event
        public event commandDelegate commandEvent;

        //Attribut **** Attribut

        private int Quantite ;

        //Constructeur ***** Constructor

        public Article(int Quantite )
        {
            this.Quantite  = Quantite ;
        }


        //Propriété ****** Property

        public int QuantiteProperty
        {
            get {
                return Quantite ;
                }
            set {
                Quantite  = value;
                }
        }
        //Méthodes **** Methods
        public void Commande(int NbrArticle)
        {
            if (NbrArticle > this.Quantite) commandEvent();
            this.Quantite  = Quantite  - NbrArticle;
        }


        public void messagaEvent()
        {
            Console.Write("Rupture de stock !!!!");
        }
    }
```

```
    public delegate void commandDelegate();

    static void Main()
    {
        Article article = new Article(20);

     article.commandEvent += new commandDelegate(article.messagaEvent);

        article.Commande(60);

        Console.Write(article.QuantiteProperty);

        Console.ReadKey();
    }
}
```

إذا لقيت أدنى صعوبة في المثال فلا تتردد في إعادة قراءة الفقرة من الأول بتمعن فالأحداث جزء لا يتجزأ من البرمجة المتطورة.



22. الإجراءات المجهولة Les méthodes anonymes (Anonymous methods)

ظهر مفهوم الإجراءات المجهولة مع ظهور c# 2.0 ، وأهميتها تكمن في السماح بإنشاء الإجراء و تمريره للمفوض في نفس الوقت .

وهي شبيهة بما كنا قد عملناه سابقا مع الأحداث، و لتقريب المفهوم بكثرة سنعرض المثال نفسه باستعمال حدث متصل بإجراء عاد مرة و بإجراء مجهول مرة أخرى، كي تسهل المقارنة و نستسيغ المفهوم بإذن الله.

أما المثال فسيكون عبارة عن حدث يتم إطلاقه عند تجاوز نسبة ساعات الغياب القصوى، سننشئ فئة نسميها Absence وسنعمل عليها لتضمين هذا المفهوم:

```
class Absence
{
    int Hours;

    public int getHours
    {
        get { return Hours; }
        set { Hours = value; }
    }

    public Absence(int hours)
    {
        this.Hours=hours;
    }

    public void verifyAbsence(int MaxHours)
    {
        if (getHours > MaxHours) Program.onExceed();
    }

 public   static void alertEvent()
    {
        Console.WriteLine("Avertissement !!");
    }
}
```

تضم الفئة Absence متغيرا داخليا وحيدا و هو عدد الساعات المتغيبة Hours، كما تضم مشيدا و خاصية و دالة اسمها verifyAbsence تقارن قيمة الساعات المررة وهي القيمة العليا مع قيمة الساعات المتغيبة، فإذا كانت الخاصية getHours وهي بمثابة المتغير hours أكبر من القيمة المسموح بها تم إطلاق الحدث onExceed.



وهنا بقية الكود :

```
    public delegate void myDelegate();
    static event myDelegate onExceed;
    static void Main(string[] args)
    {
        onExceed += new myDelegate(Absence.alertEvent);
        Absence absence = new Absence(10);
        absence.verifyAbsence(8);
        Console.Read();
    }
}
```

كما رأينا سابقا، قمنا بالإعلان عن المفوض و عن الحدث ثم قمنا بربط الحدث onExceed بالدالة alertEvent التي تمكنا من الولوج إليها عن طريق اسم الفئة لأنها ثابتة static . و في الأخير أعلنا عن كائن من الفئة Absence و مررنا إليه قيمة الساعات المتغيية وهي 10 ثم نادينا على الدالة verifyAbsence وأعطيناها القيمة 8 و هي القيمة القصوى المسموح بها و بالتالي سيتم تنفيذ الحدث و النداء على الإجراء alertEvent .

جرب قيما أخرى و سترى بأن الحدث لا يستجيب إلا إذا كانت قيمة الساعات المتغيبة أكبر من قيمة الساعات القصوى الممررة عبر الإجراء verifyAbsence ، يعتبر هذا المثال تدعيما لما رأيناه سابقا مع الأحداث، حاول أن تنسج أمثلة أخرى على غراره وأن تطورها و تقوم بربطها بالمفاهيم التي رأيناها سابقا حتى تألفها.

لا ترتبك فالأمر يتعلق بنفس السيناريو السابق و إن لم تتمكن من استيعابه فارجع خطوة إلى الخلف و راجع الأحداث.

إلى الآن لم نضف شيئا جديدا، سنستغل نفس المثال لنسرد من خلاله الإجراءات المجهولة،فما عليك سوى التركيز :

```
class Absence
{
    int Hours;

    public int getHours
    {
        get { return Hours; }
        set { Hours = value; }
    }
    public Absence(int hours)
    {
        this.Hours=hours;
    }
    public void verifyAbsence(int MaxHours)
    {
        if (getHours > MaxHours) Program.onExceed();
    }
}
```

كما ترى لقد استغنينا عن الإجراء alertEvent لأننا سنستعملها مباشرة عند تمريرها للحدث، وهنا تكمن قيمة الإجراءات المجهولة.والآن تابع ما يلي :

```
public delegate void myDelegate();
static event myDelegate onExceed;
static void Main(string[] args)
{

    onExceed += delegate()
    {
        Console.WriteLine("Avertissement !!");
    };

    Absence absence = new Absence(10);
    absence.verifyAbsence(8);
    Console.Read();
}
```

هذا كل شيئ ، فقط عند ربط الحدث بالإجراء نضع **delegate** بدل اسم المفوض الأصلي ثم نفتح القوس و نضيف متغيرات داخلية إذا كان المفوض ينتظر متغيرات داخلية ، أما في حالتنا هذه فهو فارغ.



ثم نضع كود الإجراء بين اللامتين، الأمر سهل جدا جرب و سترى.

وهنا سأضع الكود كاملا حتى نعود إليه إذا استعصى علينا أمر ما:

```
class Program
{
    class Absence
    {
        int Hours;

        public int getHours
        {
            get { return Hours; }
            set { Hours = value; }
        }
        public Absence(int hours)
        {
            this.Hours=hours;
        }
        public void verifyAbsence(int MaxHours)
        {
            if (getHours > MaxHours) Program.onExceed();
        }
    }


    public delegate void myDelegate();
    static event myDelegate onExceed;
    static void Main(string[] args)
    {

        onExceed += delegate()
        {
            Console.WriteLine("Avertissement !!");
        };

        Absence absence = new Absence(10);
        absence.verifyAbsence(8);
        Console.Read();
    }
}
```

23. العبارات لامدا Les expressions lambda(lambda expressions)

صدقني إن قلت لك أني خفت من هذا الإسم عند قراءته أول مرة و كأنه يضم بين طياته شيئا من علم أجهله، و هذا ما أتوقعه منك أيضا إن كنت تراه لأول مرة، ولكنني تفاجأت عندما اطلعت عليه نظرا لسهولته و سلاسته و هذا ما ستجده أيضا.

العبارات لامدا هي شكل آخر من الإجراءات المجهولة لا أقل و لا أكثر، ظهرت مع C# 3.0 لتسمح للمبرمج من استعمال الدوال و الإجراءات و إنشاءها في وقت واحد، وتختلف عن الإجراءات المجهولة في الشكل فقط حيث سنستعمل الرمز => بعد المتغير الداخلي Parameter للمفوض أي ما يمكن اختزاله في الصيغة التالية:

```
(Parameter) => Instruction;
```

حيث Parameter هو المتغير الداخلي و Instruction هي الأوامر التي نرغب في تنفيذها.

وحتى لا نذهب بعيدا سنحاول أن نستعمل نفس المثال السابق و نضمنه العبارات لامدا.

سنحافظ على نفس الفئة و ستلاحظ أن التغيير طفيف، حاول أن تقارن هذا المثال بالسابق :

```
public delegate void myDelegate();
static event myDelegate onExceed;
static void Main(string[] args)
{
    onExceed = () => Console.WriteLine("Avertissement");
    Absence absence = new Absence(10);
    absence.verifyAbsence(8);
    Console.Read();
}
```

السطر الأصفر هو كيفية استعمال العبارات لامدا، الأقواس فارغة لأن مفوضنا لا يحتوي على متغيرات داخلية، نفذ البرنامج و ستصل إلى نفس النتائج السابقة بإذن الله.

هناك استعمالات أخرى للعبارات لامدا، ولكنها تعتمد على نفس الصيغة، سنحاول فقط سرد مثال عاد غير متعلق لا بمفوضات و لا بأحداث حتى تتمكن من أخذ رؤية شمولية حول هذا المفهوم.



سننشئ لائحة <>List من نوع نصي و سنضع بها أسماء الخلفاء الراشدين رضوان الله عليهم ثم سنظهرهم بالإعتماد على العبارات لامدا :

```
class Lambda
{
    static void Main(string[] args)
    {
        List<String> Alkholafaa = new List<string>
        { "Abu bakr", "Omar", "Ali", "Othman" };
        Console.WriteLine("Alkholafaa arrachidoun are :\n");
        Alkholafaa.ForEach(Item =>
            {
                Console.WriteLine("\t-" + Item);
            });
        Console.Read();
    }
}
```

كان بإمكاننا فعل ذلك بطريقة أسهل و لكن قمنا بذلك فقط لننوع الطرق و لنتعرف أكثر على العبارات لامدا.

بالنسبة لForEach فهي دالة لها نفس دور الأمر foreach الذي رأينا سابقا.



الخاتمة:

الحمد لله أولا وأخيرا والصلاة و السلام على أشرف المخلوقين، الآن يا أخي كتب لنا أن نفترق قليلا إلى أن نلتقي قريبا إن شاء الله في جزء جديد.

لن أقول لك بأننا تعلمنا كل شيء عن السي شارب ولكنني سأهنؤك لأنك حققت الأهم بتعلمك أساسيات هذه اللغة و مفاهيمها المتطورة ، لأن الآتي أسهل بكثير مما رأينا في هذا الكتاب المتواضع.

لن أغضب إذا انتقدت أسلوبي في الشرح أو اقترحت علي فكرة جديدة أو نصحتني بشيء ما أو نبهتني إلى خطإ في الكتاب بل سأكون مدينا لك لأنك ساعدتني.

أنا لا أبرئ نفسي من الخطأ فالعصمة للأنبياء عليهم السلام أما عبد ربه فلا يرى أكثر مما تمتد إليه عيناه، سامحني إن لم أشرح مفهوما ما جيدا و لا تنس أن تبعث إلي بكل شيء إلا المال إلى بريدي التالي :

Khalid_Essaadani@Hotmail.Fr

وجزاك الله خيرا





المراجع

- **Des livres anglais :**
  - Essential Csharp 3 For dot NET Framework 3.5
  - DotNet 3.5 Pro C# 2008 and the .Net 3.5 Platform
  - CSharp Module 12 Operators Delegates and Events
  - Programming CSharp, 4th Edition
  - Pro Csharp with NET3[1].0 Special Edition Apress
  - Manual Visual CSharp 3.0 And Visual Studio 2005

- **Des livres français :**
  - APPRENTISSAGE DU LANGAGE C# 2008 et du Framework .NET 3.5
  - Csharp - L'essentiel en concentré (dotnet france)
  - CSharp Language Specification
  - Notions avancées du langage Csharp (dotnet france)
  - Notions fondamentales du langage Csharp (dotnet france)

- **Des livres arabes :**
  - المرجع المبسط في البرمجة الكائنية التوجه (أحمد نجم)
  - خطوة بخطوة مع الفيجوال ستوديو 2008 (أحمد جمال خليفة)
  - برمجة إطار عمل الدوت نيت باستخدام الفيجوال بسيك (تركي العسيري)

- **Des sites web :**
  - http://msdn.microsoft.com
  - http://www.vb4arab.com
  - http://www.fabienrenaud.com

اللهم اجعله عملا خالصا لوجهك

إلى لقاء قريب إن شاء الله